



第三届 eBPF开发者大会

[www.ebpftravel.com](http://www.ebpftravel.com)

# mmpf: Android系统上基于eBPF的性能观测实践

荣耀终端 | OS Kernel Lab

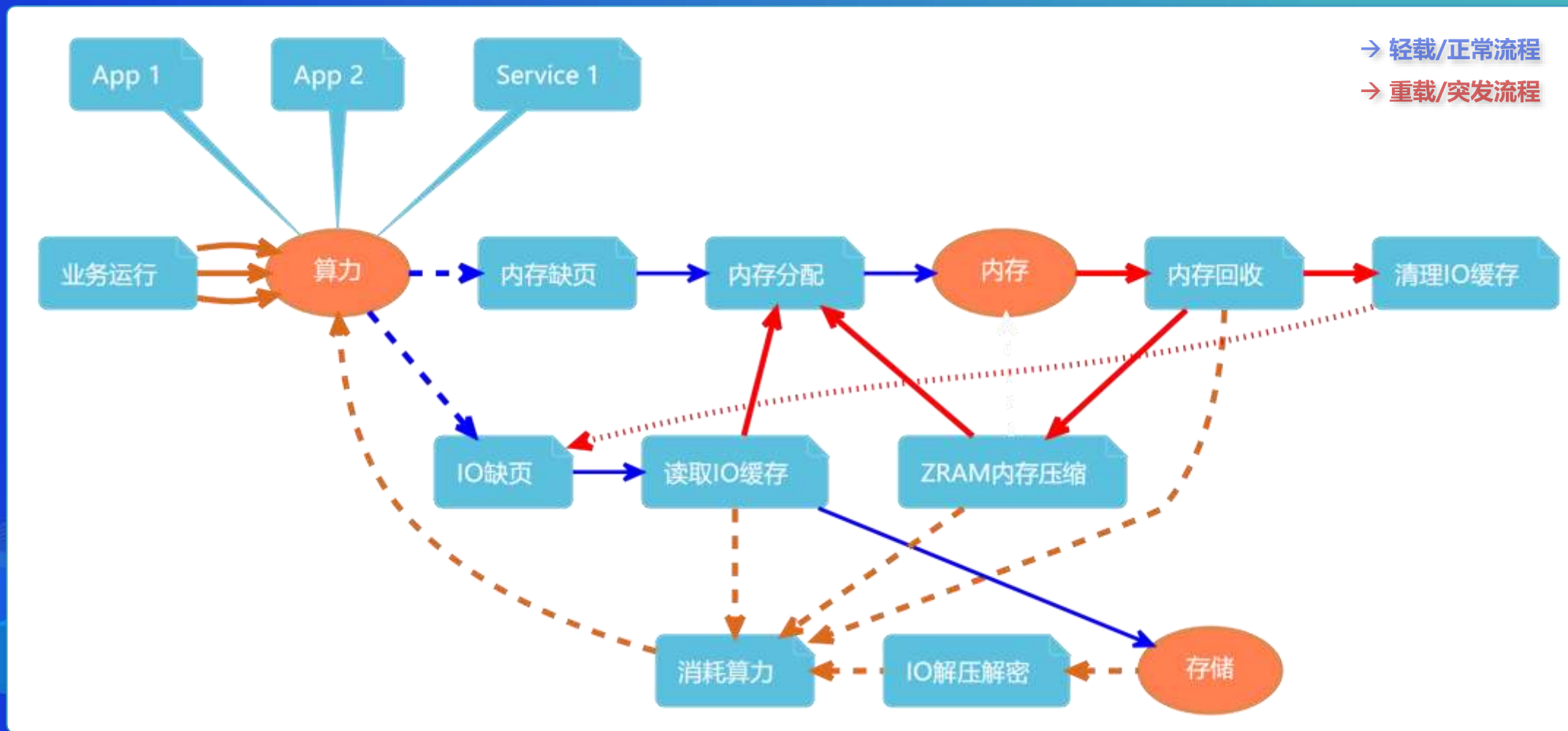
李志卫 / 夏兵

中国·西安

## 作者介绍

- 李志卫：Android系统性能优化专家，eBPF技术爱好者，有多年嵌入式系统和Android系统优化经验，乐于构建可复用的基础观测能力。
- 夏兵：性能工程专家，eBPF技术爱好者，致力于让性能分析与优化变得更简单。在系统性能可观测、性能剖析与自动化诊断等领域有多年的实战经验。

## Android系统性能背景：用户使用场景丰富，硬件资源受限



- 硬件资源受限的条件下：如何满足复杂多变的用户诉求？如何观测瓶颈点？
- 现有观测工具：trace / perfetto / simpleperf / ...

# 目录

需求分析

设计开发

成果展示

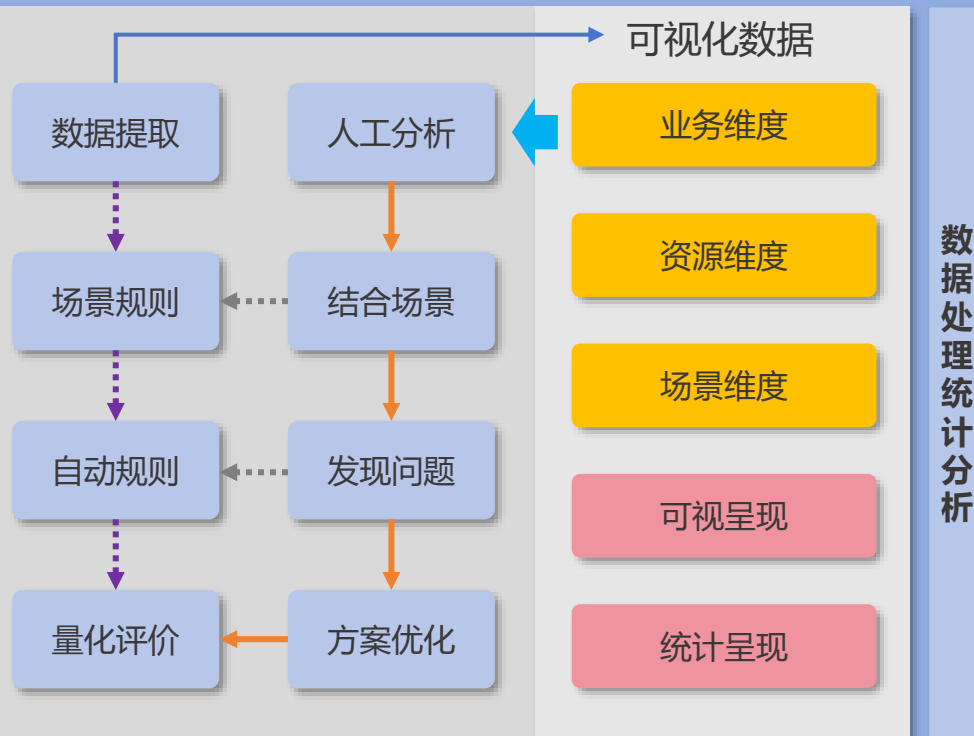
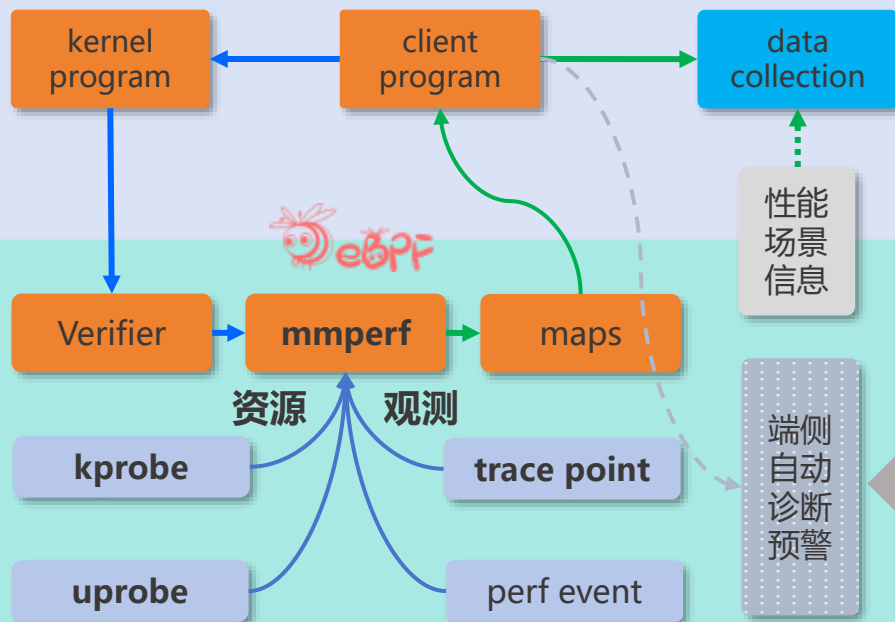
经验分享

端侧

离线数据分析

应用空间

内核空间



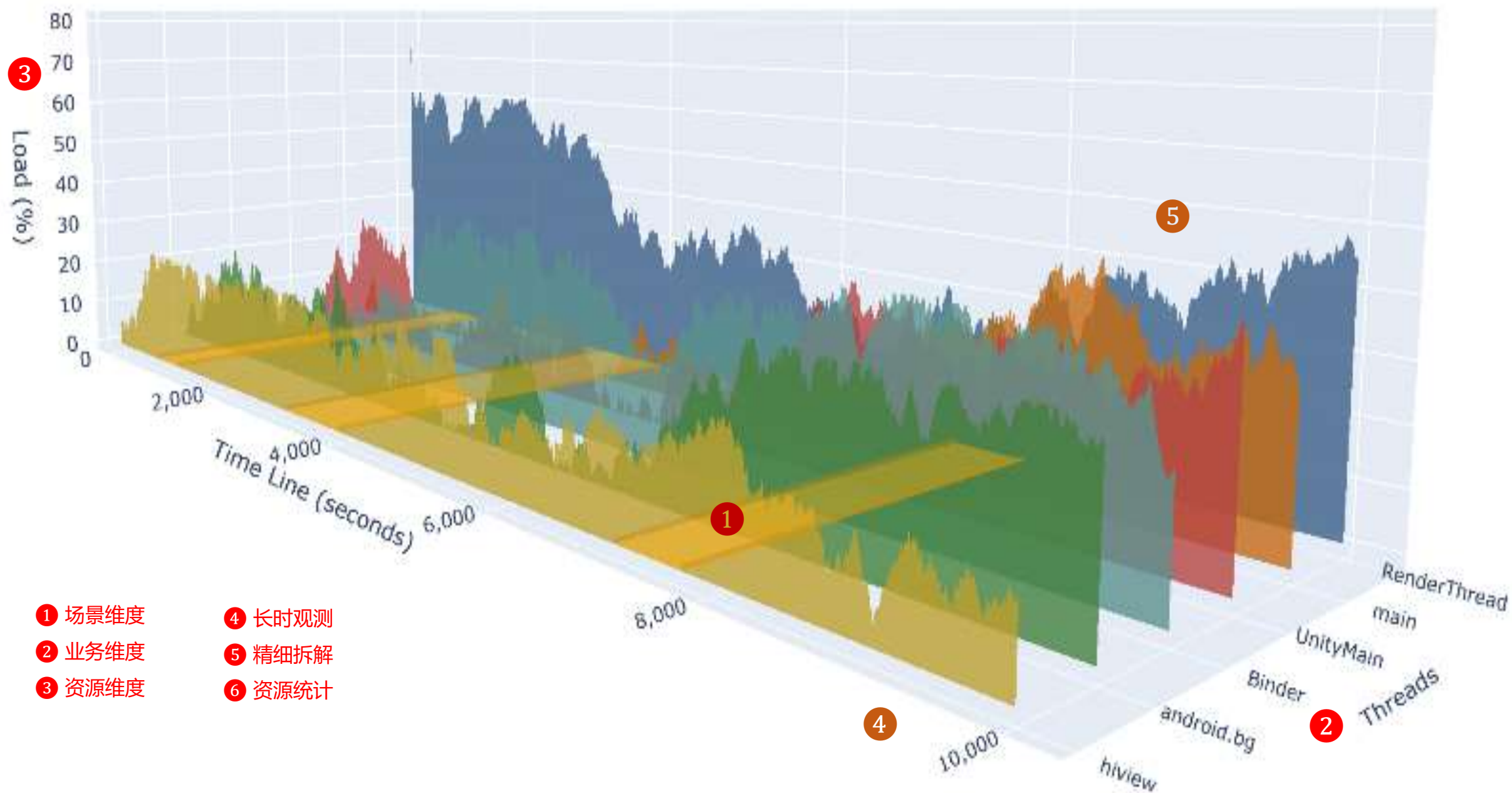
mmperf框架

# mmperv需求来源：Android性能观测诉求收集





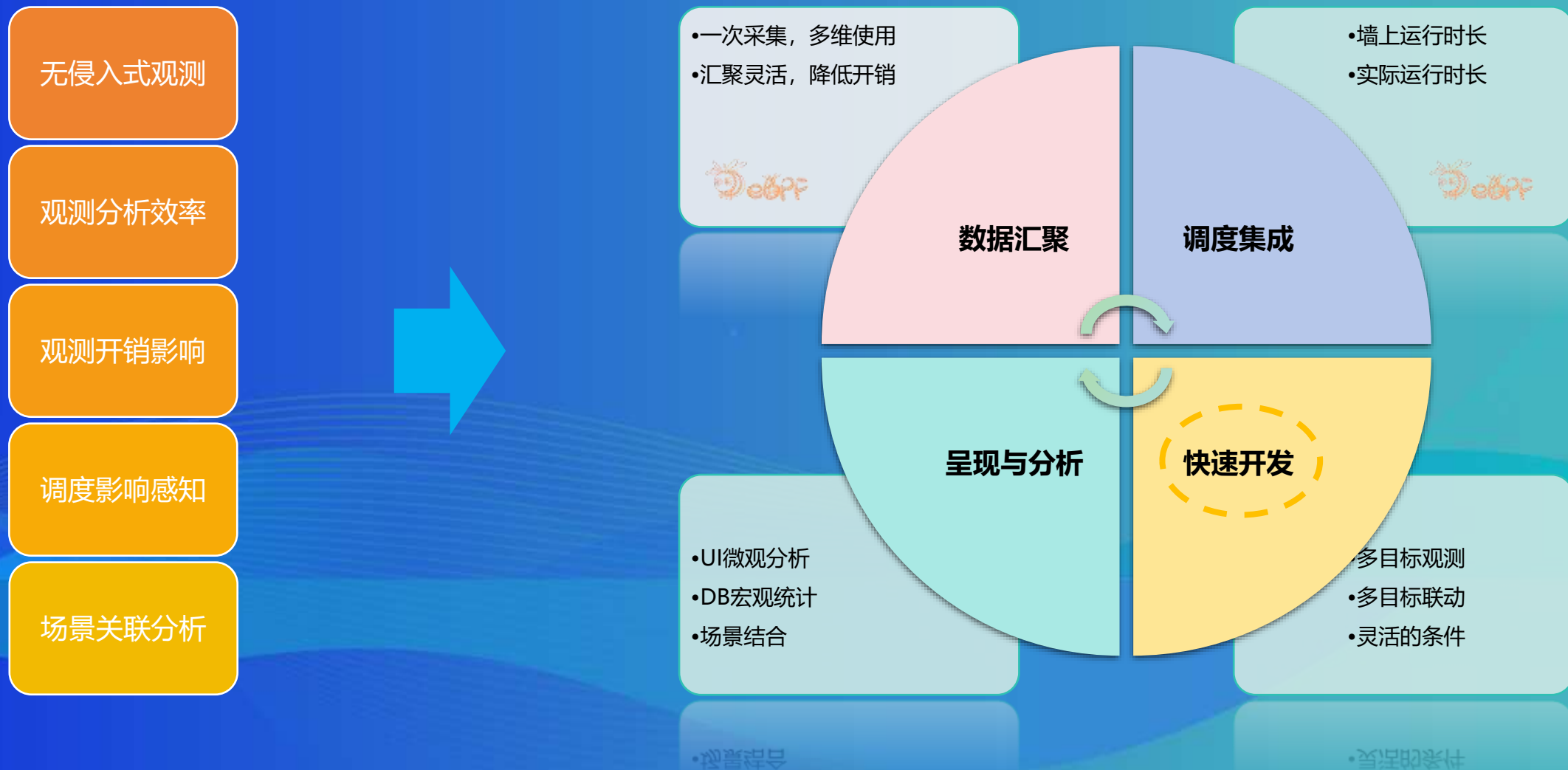
# mmperf需求设计：多维度分解，长时间观测，概率学统计



# mmperv设计目的：差异化诉求，低开发量，实用为主

维度	mmperv	simpleperf	perfetto / systrace	bcc / bpftrace / ..
设计目的	基于eBPF的耗时，计数和调用栈观测 分解到时间和任务	多功能的性能观测工具 调用栈和短时间观测为主	基于ftrace/atrace的性能观测 工具	易用的eBPF观测工具
设计能力	<ul style="list-style-type: none"><li>➢ 轻量级，<b>长时间，可商用</b></li><li>➢ 观测目标基于任务和时间轴</li><li>➢ 数据库查询接口</li><li>➢ 非实时编译</li></ul>	<ul style="list-style-type: none"><li>➢ 周期性采集调用栈，PMU等相关信息</li><li>➢ 形成record或者status的汇聚。</li><li>➢ 无需编译</li></ul>	<ul style="list-style-type: none"><li>➢ 采集ftrace/atrace</li><li>➢ 交互式UI呈现</li><li>➢ 数据库查询接口</li><li>➢ 接入了更多的关联信息</li></ul>	<ul style="list-style-type: none"><li>➢ 采集eBPF数据</li><li>➢ 使用简单的脚本语言或者结合c语言进行数据呈现</li><li>➢ 实时编译</li></ul>
优点	<ul style="list-style-type: none"><li>➢ 灵活的观测目标与条件</li><li>➢ 灵活的<b>数据聚合</b></li><li>➢ 与用户<b>场景关联</b></li><li>➢ 结合<b>调度信息</b></li><li>➢ 有数据库查询能力支持</li><li>➢ 体积小（xxxKB）</li></ul>	<ul style="list-style-type: none"><li>➢ 采集信息丰富，包含调用栈</li><li>➢ 当前也有很好的UI工具，如firefox profiler等</li></ul>	<ul style="list-style-type: none"><li>➢ 功能丰富</li><li>➢ 高速迭代</li><li>➢ 优雅的UI呈现</li><li>➢ Android设备主要分析工具</li><li>➢ 有强大数据库查询能力支持</li></ul>	<ul style="list-style-type: none"><li>➢ 语法简洁，学习成本较高</li><li>➢ 能做很多常见的简单的数据汇聚</li></ul>
缺点	<ul style="list-style-type: none"><li>➢ 目的性强，功能单一</li><li>➢ 信息不如现有工具丰富</li><li>➢ <b>UI简陋，未来考虑接入perfetto</b></li></ul>	<ul style="list-style-type: none"><li>➢ 数据量大，开销高</li></ul>	<ul style="list-style-type: none"><li>➢ 数据量较大</li></ul>	<ul style="list-style-type: none"><li>➢ 依赖编译环境</li><li>➢ Android系统默认无法直接使用</li></ul>
应用难度	<ul style="list-style-type: none"><li>➢ 低*</li></ul>	<ul style="list-style-type: none"><li>➢ 低</li></ul>	<ul style="list-style-type: none"><li>➢ 低</li></ul>	<ul style="list-style-type: none"><li>➢ 低</li><li>➢ Android系统部署难度中等</li></ul>
建议场景	<ul style="list-style-type: none"><li>➢ <b>长时间观测指定目标，进行概率统计与分析</b></li><li>➢ <b>需要基于时间和任务的分解与聚合</b></li><li>➢ <b>需要量化观测目标与用户场景的关系</b></li></ul>	<ul style="list-style-type: none"><li>➢ 短时间观测指定目标的调用栈，PMU等信息</li></ul>	<ul style="list-style-type: none"><li>➢ 短时间观测性能场景（现有trace信息可满足分析诉求）</li><li>➢ 复杂的综合因素分析</li></ul>	<ul style="list-style-type: none"><li>➢ 有明确的自定义观测目标。</li><li>➢ 可复现的故障现场</li></ul>

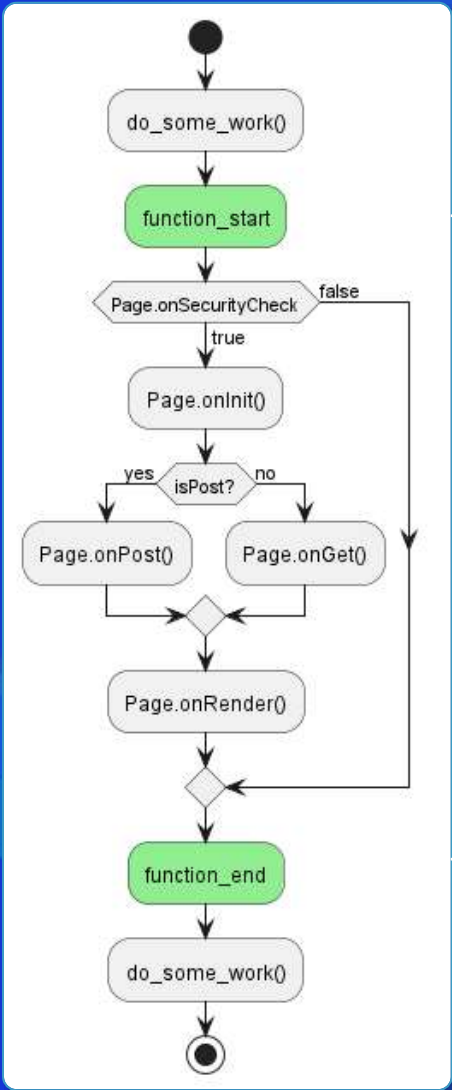
# mmperrf设计目标：抽象为公共框架以便快速开发



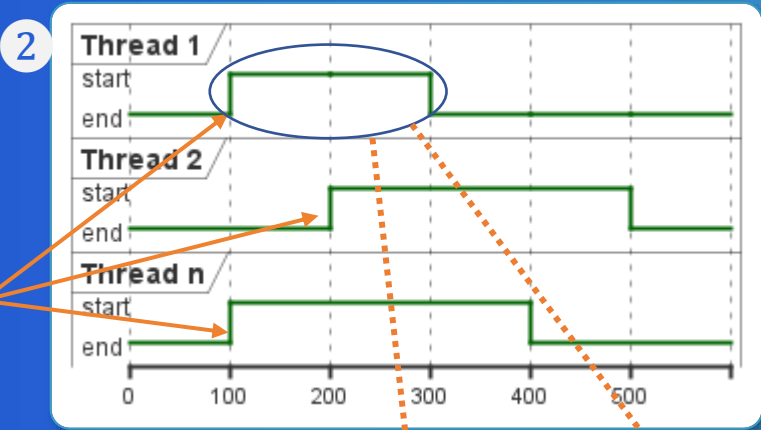


# mmp perf中能力的抽象与设计：以运行时长统计为例

开发视图



运行视图

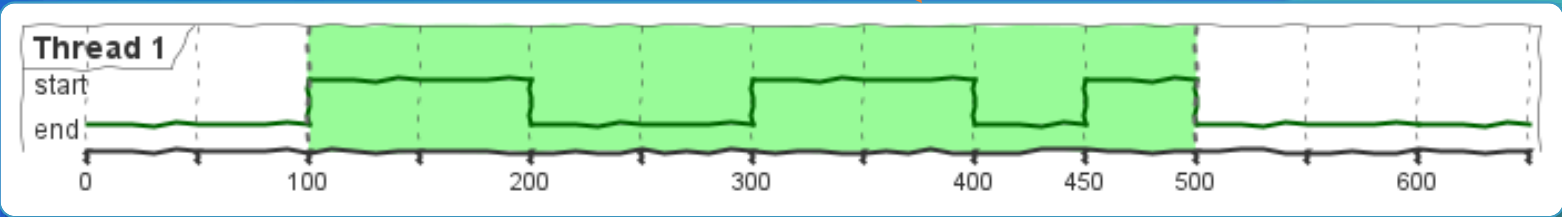


1

观测目标

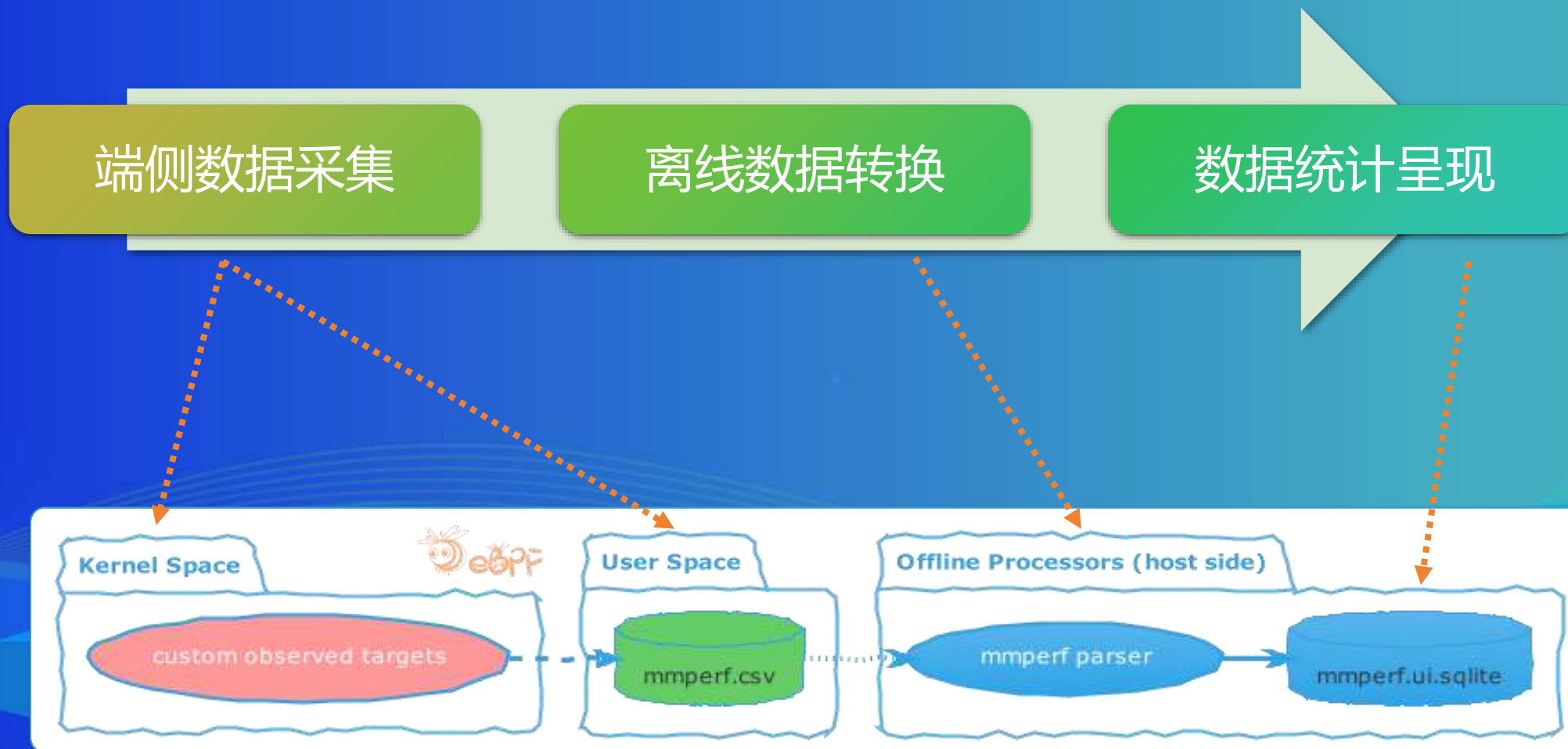


数据汇聚

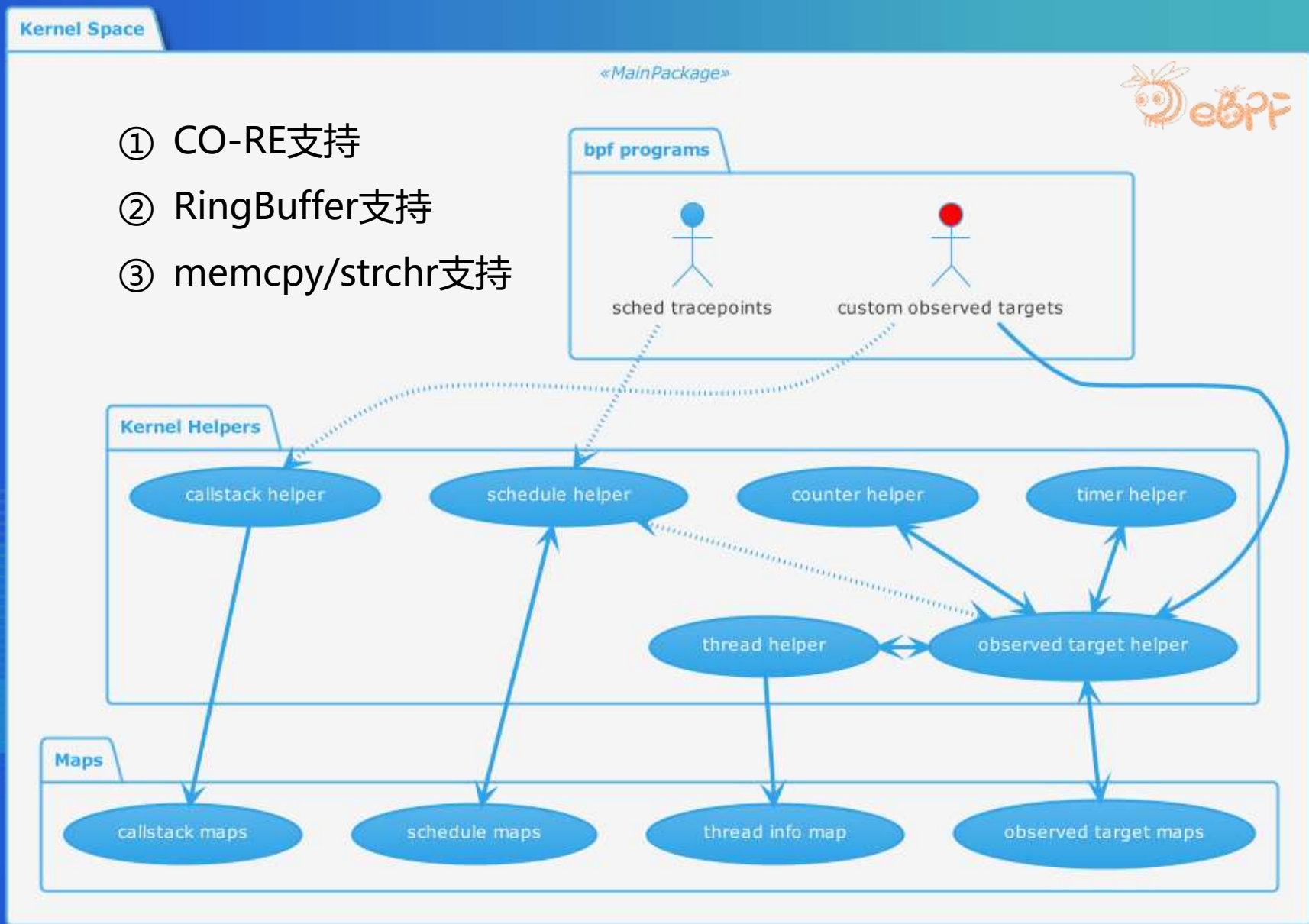


- ① 抽象观测目标
- ② 多任务并行记录
- ③ 提取调度信息
- ④ 数据内核汇聚

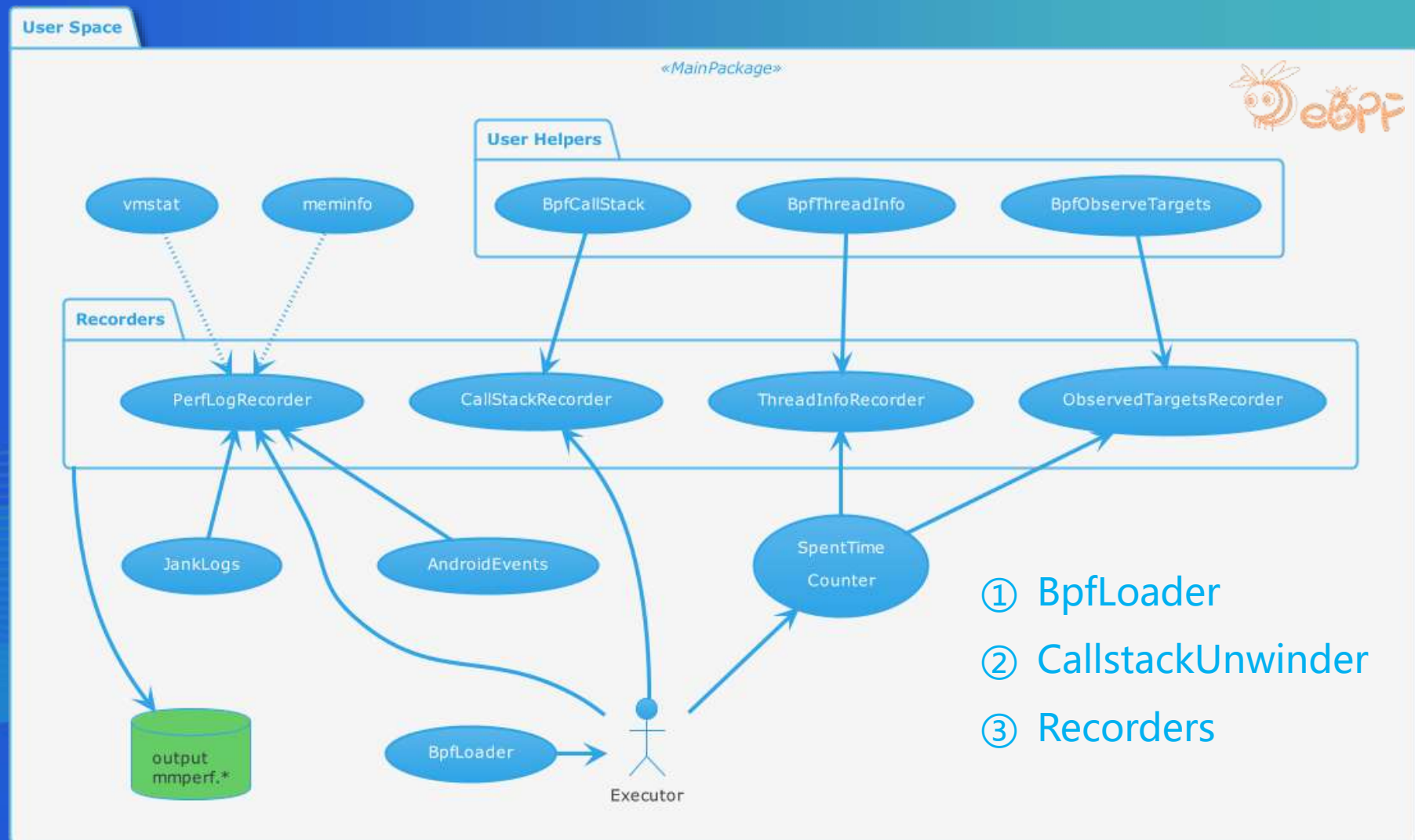
## mmp perf 分层设计：充分借助 eBPF 高效灵活的数据采集优势



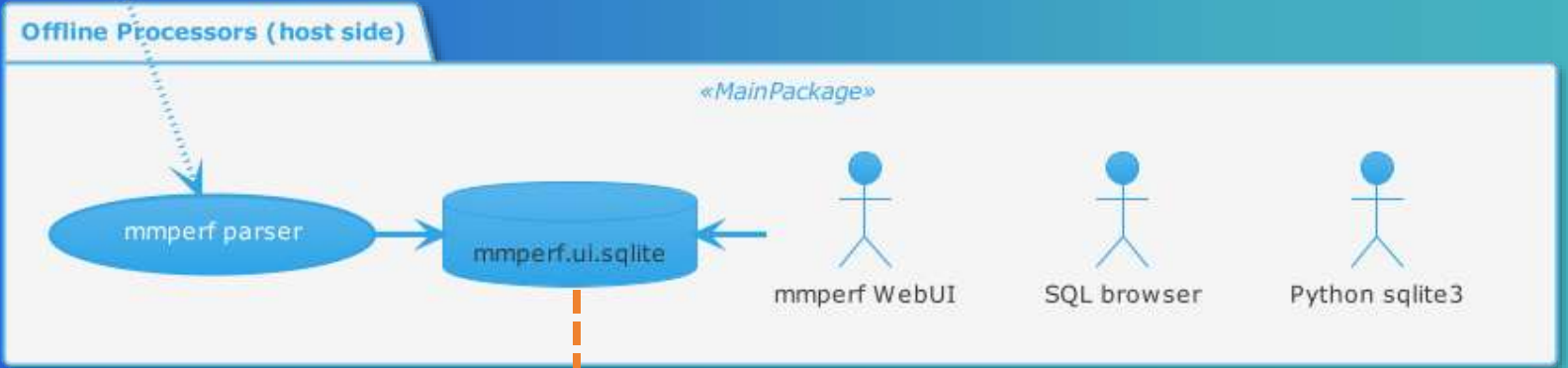
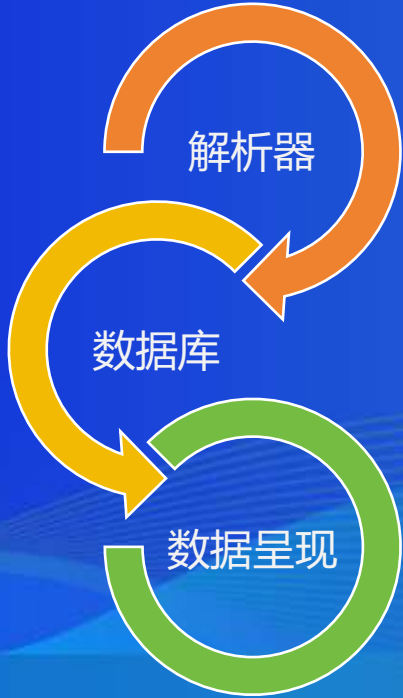
# mmp perf 分层设计：内核态



# mmp perf 分层设计：用户态



# mmperv分层设计：数据处理

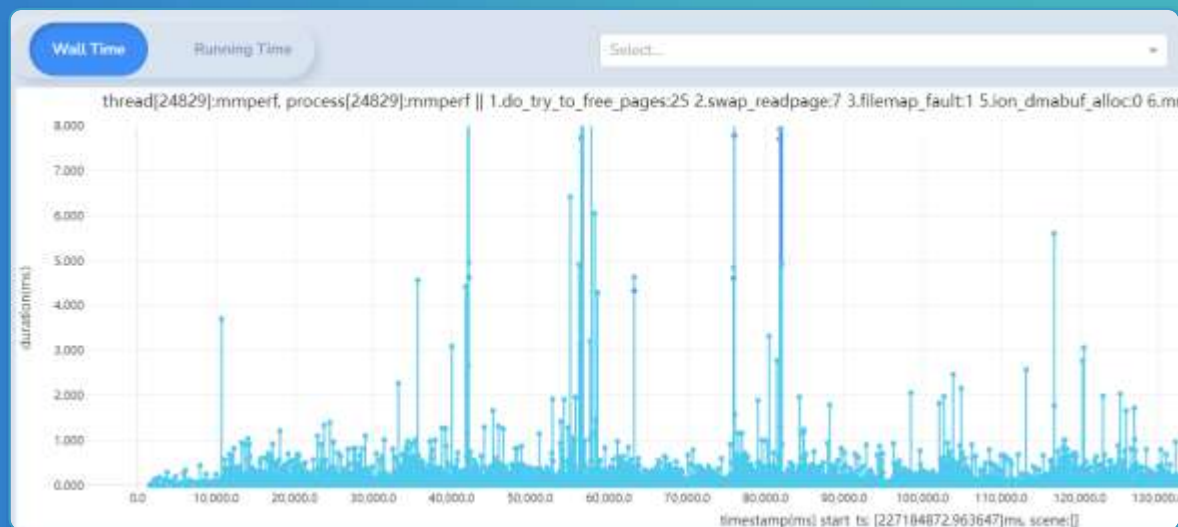


Threads			Processes			Application			Scenes				
process			total						do_try_to_free_pages				
process	↕threads	↕uapid	↕ count	↕ dur_wall	↕ dur_running	↕idle_count	↕prio_min	↕prio_max	↕count	↕ dur_wall	↕dur_running	↕idle_count	↕
	Aa	Aa	Aa	Aa	Aa	Aa	Aa	Aa	Aa	Aa	Aa	Aa	
rdware.camera.provider@2.4-service_64	248	114	352503	9,094,191,433	4,334,476,541	6659	98	120	767	1,725,331,495	1,023,878,734	980	
lor.qti.hardware.display.allocator-service	4	41	386619	5,313,409,363	4,150,631,397	1781	97	100	291	343,808,119	278,487,224	158	
system_server	187	6	410473	5,444,221,748	2,625,070,914	2630	89	130	149	662,639,015	286,828,964	249	
mmperv	1	1	99569	1,499,916,442	886,350,973	493	117	117	100	293,202,080	114,778,274	131	
com.hihonor.camera	101	72	39975	1,336,876,690	472,189,240	418	97	130	94	515,368,747	132,833,586	89	
[zswapd0]	1	0	25552	1,306,357,795	722,086,636	170	135	135	81	918,088,387	349,012,023	113	
com.qiyi.video	553	98	342715	18,460,585,010	2,689,037,145	4056	100	139	71	173,513,179	88,650,471	64	
com.autonavi.minimap	667	106	352209	8,625,048,332	2,383,308,456	2131	100	139	35	246,133,543	110,670,000	88	
/system/bin/logd	6	12	10664	1,000,366,348	246,790,417	120	130	130	25	412,208,128	76,040,474	33	
/system/bin/hiview	12	47	8697	607,881,767	229,442,231	162	120	130	22	150,083,072	54,665,364	42	
826	826	826	6303065	142,500,746,281	50,719,458,346	51359	0	139	1986	7,547,131,806	3,156,198,745	2536	1



# mmp perf实现效果：表格，时间线图，统计学数据，场景关联

Statistics				Threads		Processes				Application			
thread										total			
tid	thread	pid	process	is_main	upid	upid	count	dur_wall	dur_running	idle_c			
1211	allocator-serv	1211	/vendor/bin/hw/vendor.qti.hardware.display.allocator-service	1	63	806	145267	2,134,588,608	1,739,833,400				
1095	provider@2.4-se	1095	/vendor/bin/hw/android.hardware.camera.provider@2.4-service_64	1	733	349	46266	719,453,111	479,292,243				
24829	mmp	24829	mmp	1	3	204	95069	1,499,916,442	886,150,973				
199	zswapd0	199	[zswapd0]	1	715	0	25532	1,306,357,795	722,086,636				
4056	HWfnder:1211_3	1211	/vendor/bin/hw/	0	63	806	84441	1,130,834,804	881,485,136				
1881	provider@2.4-se	1095	/vendor/bin/hw/andro	0	733	349	3193	547,595,228	95,617,735				
28785	Thread-61	27979		0	299	33	657	81,988,647	61,561,051				
1246	HWfnder:1211_2	1211	/vendor/bin/hw/	0	63	308	77511	1,236,117,990	864,664,293				
5437	SoundDecoder_2	5072	com.hihonor.camera	0	732	100	347	154,610,877	58,547,596				
5438	AsyncTask #1	5072	com.hihonor.camera	0	732	100	386	699,882,658	127,640,151				
12385	12385	12385		12385	12385	12385	12385	8300060	142,900,746,201	30,719,438,846			



# mmp perf实现效果：自动化处理数据，完整调用栈打印

S  
Q  
L

```
sql > query sum_appstart_spent_time_group_by_prio_max.sql
1 SELECT row_number() OVER (OR
2 st.func_id
3 f.value
4 st.prio_max
5 sum(st.count)
6 min(st.dur_wall)
7 max(st.dur_wall)
8 avg(st.dur_wall)
9 sum(st.dur_wall)
10 min(st.dur_running)
11 max(st.dur_running)
12 avg(st.dur_running)
13 sum(st.dur_running)
14 max(st.prio_max)
15 min(st.prio_min)
16 FROM view_spent_time st
17 LEFT JOIN
18 function f ON st.func_id
19 WHERE (st.scenes & (1 << 9)
20 GROUP BY st.func_id,
21 st.prio_max
```

测试模型中自动化的获取统计数据

J  
A  
V  
A

C  
+  
+

C











```
__start_thread+0x40 (/apex/com.android.runtime/lib64/bionic/libc.so)
__pthread_start(void*)+0xd0 (/apex/com.android.runtime/lib64/bionic/libc.so)
art::Thread::CreateCallback(void*)+0x57c (/apex/com.android.art/lib64/libart.so)
java.lang.Thread.run+0x50 (/system/framework/arm64/boot.oat)
java.lang.Daemons$Daemon.run+0xb4 (/system/framework/arm64/boot-core-libart.oat)
java.lang.Daemons$FinalizerDaemon.runInternal+0x148 (/system/framework/arm64/boot-core-libart.oat)
java.lang.Daemons$FinalizerDaemon.processReference+0x2b0 (/system/framework/arm64/boot-core-libart.oat)
android.hardware.HardwareBuffer.finalize+0x11c (/system/framework/arm64/boot-framework.oat)
libcore.util.NativeAllocationRegistry$CleanerRunner.run+0x40 (/system/framework/arm64/boot-framework.oat)
sun.misc.Cleaner.clean+0x68 (/system/framework/arm64/boot.oat)
libcore.util.NativeAllocationRegistry$CleanerThunk.run+0x4c (/system/framework/arm64/boot-framework.oat)
destroyWrapper(GraphicBufferWrapper*)+0x20 (/system/lib64/libandroid_runtime.so)
android::RefBase::decStrong(void const*) const+0x6c (/system/lib64/libutils.so)
android::GraphicBuffer::~~GraphicBuffer()+0x10 (/system/lib64/libui.so)
android::GraphicBuffer::~~GraphicBuffer()+0x48 (/system/lib64/libui.so)
android::GraphicBuffer::free_handle()+0x30 (/system/lib64/libui.so)
android::GraphicBufferMapper::freeBuffer(native_handle const*)+0x2c (/system/lib64/libui.so)
android::Gralloc4Mapper::freeBuffer(native_handle const*) const+0x3c (/system/lib64/libui.so)
android::hardware::graphics::mapper::V4_0::BsMapper::freeBuffer(void*)+0x64 (/system/lib64/libui.so)
vendor::qti::hardware::display::mapper::V4_0::implementation::QtiMapper::freeBuffer(void*)+0x10 (/system/lib64/libui.so)
gralloc::BufferManager::ReleaseBuffer(qtigralloc::private_handle_t const*)+0x15c (/vendor/lib64/libgralloc.so)
gralloc::BufferManager::FreeBuffer(std::__1::shared_ptr<gralloc::BufferManager::Buffer>)+0x10 (/system/lib64/libui.so)
gralloc::DmaManager::FreeBuffer(void*, unsigned int, unsigned int, int, int)+0x9c (/vendor/lib64/libgralloc.so)
close+0x10 (/apex/com.android.runtime/lib64/bionic/libc.so)
android_fdsan_close_with_tag+0x32c (/apex/com.android.runtime/lib64/bionic/libc.so)
__close+0x8 (/apex/com.android.runtime/lib64/bionic/libc.so)
do_notify_resume+0x28c (kernel)
task_work_run+0xcc (kernel)
__fput+0x10 (kernel)
__fput+0x180 (kernel)
dput+0x9c (kernel)
dentry_kill+0xbc (kernel)
dma_buf_release+0x0 (kernel)
```

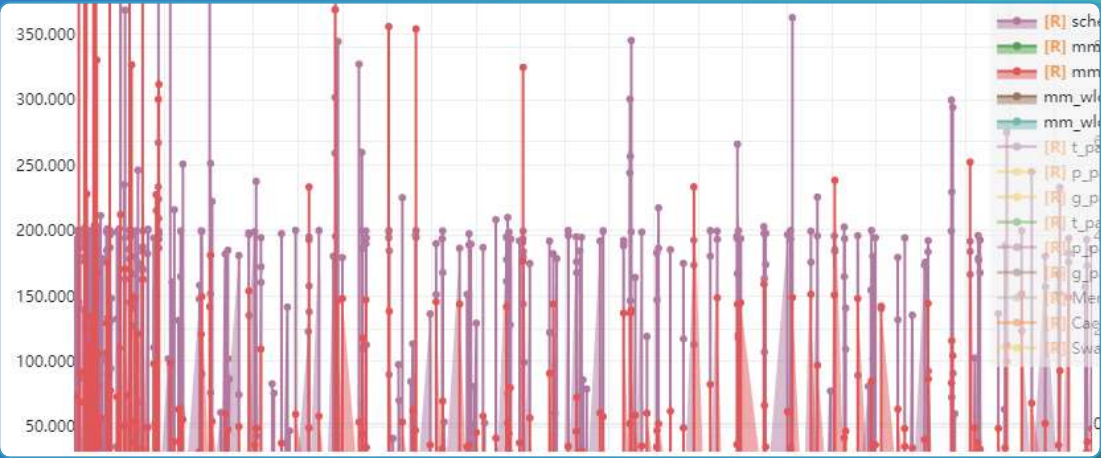
内核态C，用户态C++，用户态Java完整调用栈

mmperf实现效果：数据汇聚，大幅降低开销，同时兼顾细节

id	value	Total count
0	sched	25,459,504
1	mm_rlock_acquire	11,283,689
2	mm_rlock_dur	10,951,929
3	mm_wlock_acquire	3,311,331
4	mm_wlock_dur	3,310,702
5	page_alloc	32,603,015
6	page_free	32,206,023

1.2小时采集数据量约1.2亿次，存储占用~140MB

	mmperf.callstack_count.csv	Microsoft Excel ...	1 KB
	mmperf.callstack_event.csv	Microsoft Excel ...	10,309 KB
	mmperf.callstack_event.xlsx	Microsoft Excel ...	1,532 KB
	mmperf.cpuset	CPUSET 文件	3 KB
	mmperf.csv	Microsoft Excel ...	142,793 KB
	mmperf.events	EVENTS 文件	1,066 KB
	mmperf.jank	JANK 文件	160 KB
	mmperf.log	文本文档	5 KB
	mmperf.meminfo	MEMINFO 文件	15,646 KB
	mmperf.packages	PACKAGES 文件	22 KB

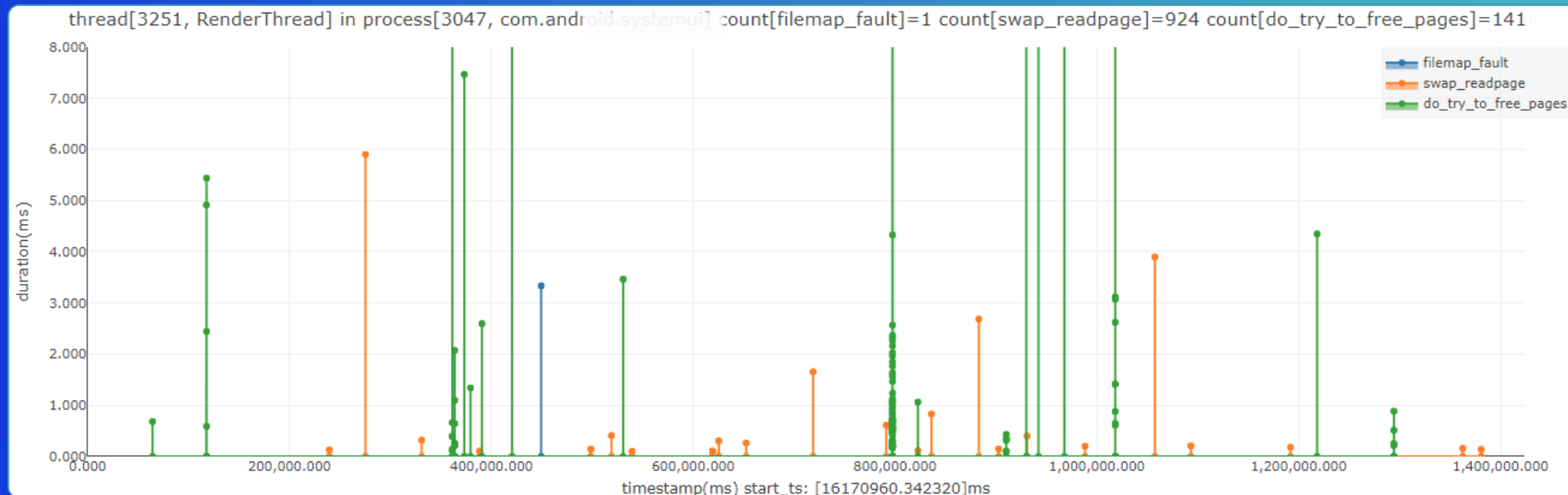


可统计，可分解

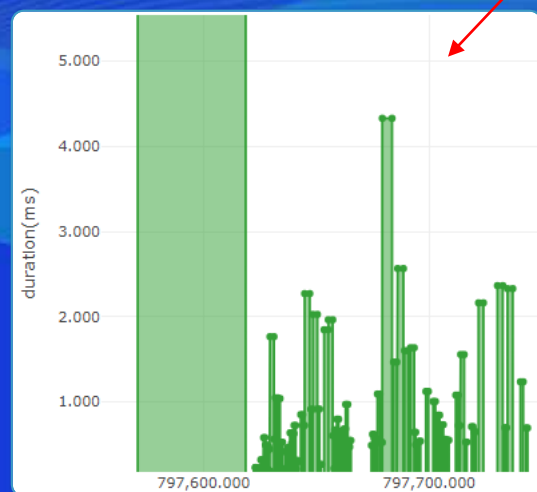


# mmp perf应用案例1：关键UI线程概率出现内存回收时间长

应用极速连续启动场景



200ms内连续慢速路径，丢24帧+

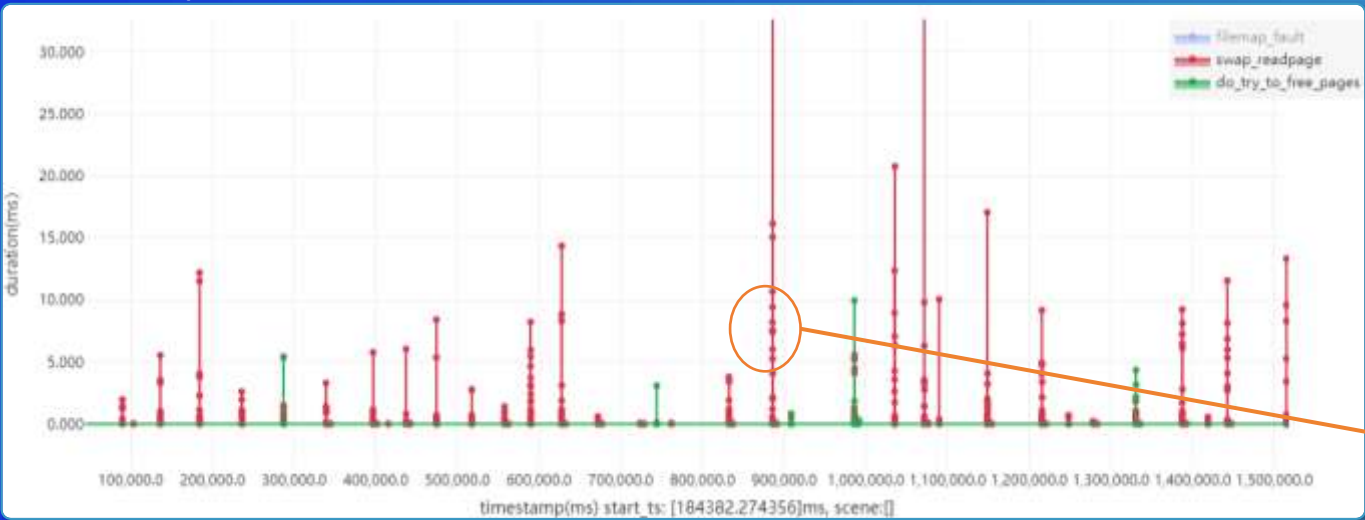


## Render关键线程案例分析：

- 重载下，小概率概率出现长耗时，文件缓存读取/ZRAM解压/慢速路径均涉及
- 其中慢速路径相对影响较大，甚至出现200ms+连续慢速路径的情况
- 按120Hz推算，丢帧影响24帧左右。

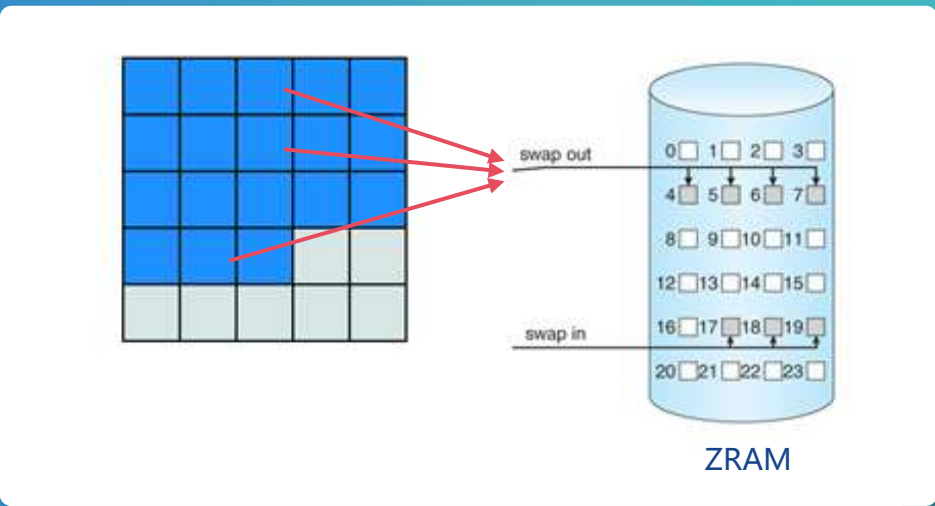
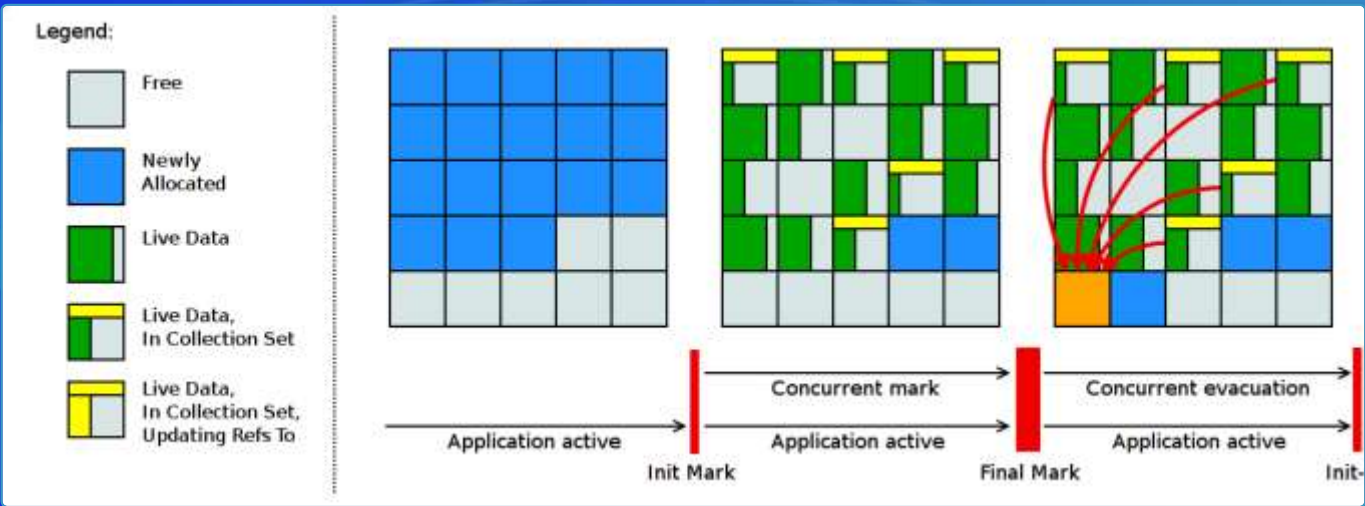
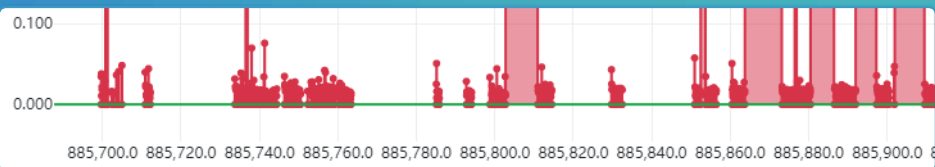
# mmp perf应用案例2：JVM GC与底层内存管理之间的视角差异冲突

thread: HeapTaskDaemon



## 视角冲突点:

- GC对象分散存储于内存页
- 内存页可能会被大量压缩
- GC过程产生数据搬运
- 搬运过程带来缺页数据换入





mmp perf应用案例3：前后台场景下资源供给的自动化统计

scene_name	func_id	func_name	app_count	process_count	thread_count	count	dur_wall	dur_running	dur_wall_fg	dur_running_fg
frame skip	1	do_try_to_free_pages	41	61	169	251	6,400,029,331	970,546,372	175,997,133	58,754,811
frame skip	2	swap_readpage	57	76	427	721	1,736,689,926	212,458,770	1,247,917	1,145,811
frame skip	3	filemap_fault	75	119	604	1924	53,188,850,918	2,177,576,477	815,670,627	103,147,711
frame skip	5	ion_dmapbuf_alloc	6	6	18	94	1,910,517,809	1,285,246,746	1,353,178	1,353,178
frame skip	6	mm_page_alloc_entry	118	313	3574	15744	18,975,740,832	5,829,254,678	1,994,692,185	1,291,953,911
5	5	5	297	575	4792	18734	82,211,828,816	10,475,083,043	2,988,961,040	1,456,354,911



应用轮巡冷启动场景：按照场景，以及前后台区分观测目标的耗时

# mmp perf实践经验典型案例1：编译优化导致的eBPF verifier校验错误问题

```
; if (__bit_array_is_set(task_filter_users, _TASK_UID_FILTER_SIZE,
UID(uid_gid))) {
12: (bf) r1 = r0
13: (67) r1 <= 32
14: (77) r1 >= 32
15: (b7) r2 = 131072
; if (index >= size) {
16: (2d) if r2 > r1 goto pc+31

from 16 to 48: R0=inv(id=2) R1=inv(id=0,umax_value=131071,var_off=
(0x0; 0x1ffff)) R2=inv131072 R6=ctx(id=0,off=0,imm=0) R7=inv(id=1)
R10=fp0
; uint32_t index = pos >> 3;
48: (bf) r1 = r0
49: (77) r1 >= 3

; return !(arr[index] & (1 << offset));
50: (57) r1 &= 536870911
51: (18) r2 = 0xffffffffc01e753008
53: (0f) r2 += r1
54: (71) r1 = *(u8 *)(r2 +0)
R0=inv(id=2) R1_w=invP(id=0,umax_value=536870911,var_off=(0x0;
0x1ffff))
R2_w=map_value(id=0,off=8200,ks=4,vs=24584,umax_value=536870911,var_of
f=(0x0; 0
x1ffff)) R6=ctx(id=0,off=0,imm=0) R7=inv(id=1) R10=fp0

invalid access to map value, value_size=24584 off=536879111 size=1
R2 max value is outside of the allowed memory range
```

```
#define _TASK_UID_FILTER_SIZE ((32768 * 4) / 8)

int __bit_array_is_set(const uint8_t* arr,
uint32_t size, uint32_t pos) {

uint32_t index = pos >> 3; ①
uint32_t offset = pos & 0x7;
if (index >= size) { ②
TRACE_E("invalid pos %u\n", pos);
return -1;
}
return !(arr[index] & (1 << offset)); ③
}
```

```
; if (__bit_array_is_set(task_filter_users, _TASK_UID_FILTER_SIZE,
UID(uid_gid))) {
; if (index >= size) { ②
16: (2d) if r2 > r1 goto pc+31 // 16: 拿位移前数据做判断
; uint32_t index = pos >> 3;
48: (bf) r1 = r0 // 48: index=pos (0xFFFFFFFF)
49: (77) r1 >= 3 ① // 49: 位移 index=(0~0xFFFFFFFF)
; return !(arr[index] & (1 << offset));
50: (57) r1 &= 536870911 // 50: 536870911 == 0x1FFFFFFF
51: (18) r2 = 0xffffffffc01e753008 // r2 = arr
53: (0f) r2 += r1 // r2 = &arr[index]
54: (71) r1 = *(u8 *)(r2 +0) // 判定存在越界读取可能，校验失败。
```

问题现象：一段正确的程序在eBPF加载过程中，出现map访问越界的校验错误

根本原因：编译优化调整了判断条件和代码位置，导致校验失败。

解决方法：访问数据前(③)增加内存屏障 `barrier_var(index)`; 问题消失。

## mmp perf实践经验典型案例2： kprobe观测陷阱

```
// irq_work.c
void irq_work_single(void *arg) ①
{
    lockdep_irq_work_enter(flags); ②
    work->func(work);
    lockdep_irq_work_exit(flags);
}

static void irq_work_run_list(struct llist_head *list)
{
    irq_work_single(work);
}

void irq_work_run(void) ③
{
    irq_work_run_list(this_cpu_ptr(&raised_list));
}
EXPORT_SYMBOL_GPL(irq_work_run);

// smp.c
static void __flush_smp_call_function_queue() ④
{
    if (type == CSD_TYPE_IRQ_WORK) {
        irq_work_single(csd);
    }
}
```

```
BVL:/ # cat /proc/kallsyms | grep -E '.* irq_work_'
0000000000000000 T irq_work_queue
0000000000000000 T irq_work_queue_on
0000000000000000 T irq_work_needs_cpu
0000000000000000 T irq_work_single ①
0000000000000000 T irq_work_run
0000000000000000 T irq_work_tick
0000000000000000 T irq_work_sync
```

```
(gdb) b *irq_work_run +0x0 ③
Note: breakpoint 1 also set at pc 0xffffffc00822c808.
(gdb) disassemble /s 0xffffffc00822c808,0xffffffc00822c9bc
Dump of assembler code from 0xffffffc00822c808 to 0xffffffc00822c9bc:
...
211     lockdep_irq_work_enter(flags); ②
212     work->func(work);
      0xffffffc00822c878 <irq_work_run+112>: ldr x8, [x0, #16]
213     lockdep_irq_work_exit(flags);
...
```

问题现象：观测目标 ① 的数量明显少于 ② 的某一部分入口。逻辑上说不通

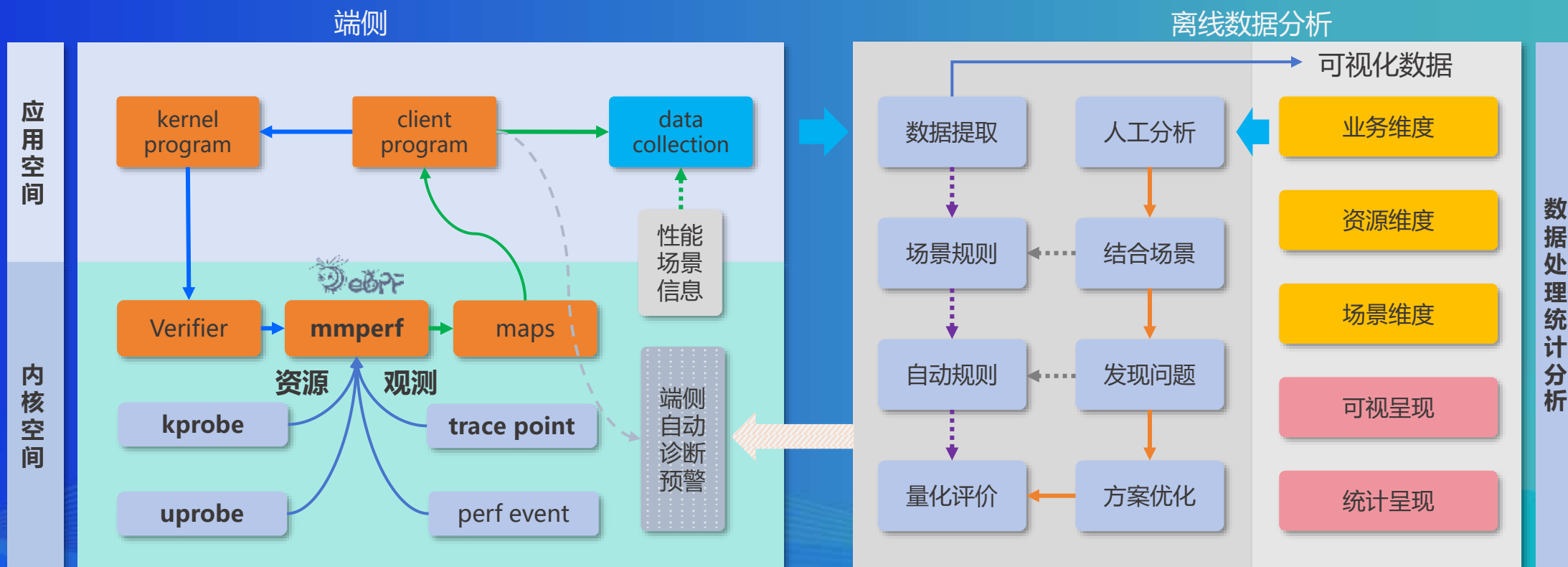
根本原因：观测目标irq\_work\_single(), 部分流程中被优化为内联。典型的kprobe使用陷阱。

解决方法：不太可能为了改善观测而修改编译参数等，规避观测目标。

影响结果：问题是隐蔽的，会引导我们得到错误甚至完全相反的结论。



# mmperrf设计总结：渐进式设计的演进原则



## ➤ 分享了一种性能观测框架设计

- ✓ **能力抽象**：耗时统计，计数统计，调用栈记录，低代码开发
- ✓ **数据聚合**：长时间观测，降低系统资源开销
- ✓ **调度结合**：界定问题的可优化方向与优化空间
- ✓ **场景关联**：场景信息导入，直观分析对场景的影响
- ✓ **见微见著**：微观与宏观并行分析（平均时延，长尾时延）

## ➤ 分享了开发中遇到的一些问题与陷阱

- ✓ eBPF程序**校验错误**问题案例
- ✓ kprobe目标的**观测陷阱**案例

## 参考链接

- **BPF Instruction Set**: <https://www.kernel.org/doc/html/v6.0/bpf/instruction-set.html>
- **simpleperf**: <https://android.googlesource.com/platform/system/extras/+refs/heads/main/simpleperf/>
- **dwarf backtrace unwind**: <https://bbs.kanxue.com/thread-274546.htm>
- **OpenJDK**: <https://wiki.openjdk.org/display/shenandoah/Main>
- **virtual memory**: [https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9\\_VirtualMemory.html](https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9_VirtualMemory.html)
- **kprobe issue on LPC2024** (Yonghong Song and Alan Maguire):  
<https://lpc.events/event/18/contributions/1945/attachments/1508/3179/Kernel%20func%20tracing%20in%20the%20face%20of%20compiler%20optimization.pdf>
- **adeb**: <https://github.com/joelagnel/adeb>
- **ExtendedAndroidTools**: <https://github.com/facebookexperimental/ExtendedAndroidTools>





**Thanks**

HONOR