第二届 eBPF开发者大会

www.ebpftravel.com

擎创的ebpf之旅

汇报人：张磊

中国·西安

目录

CONTENTS

第二届 eBPF开发者大会

www.ebpftravel.com

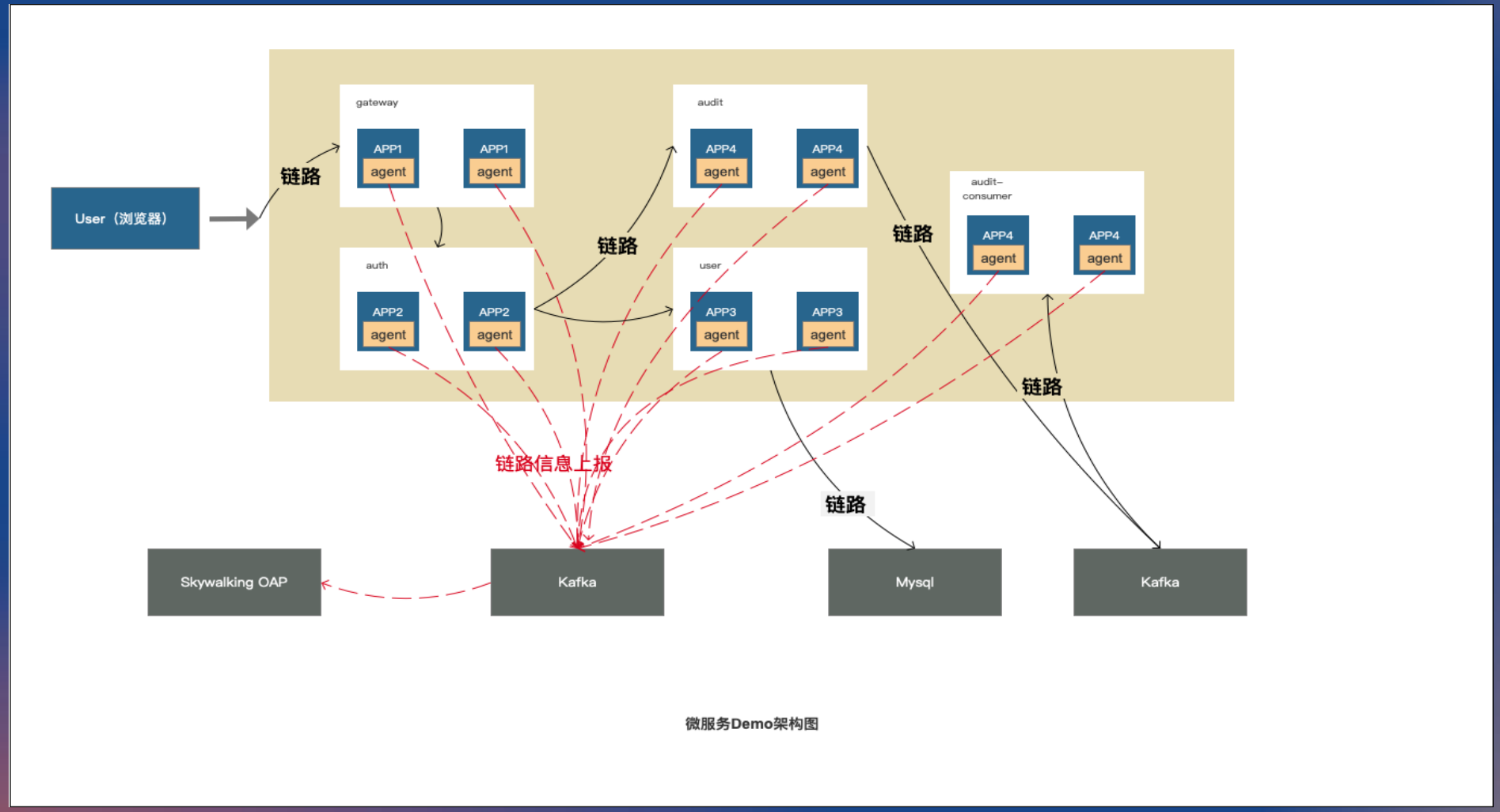
一 网络数据收集

中国·西安

① 传统网络拓扑图

传统的网络拓扑图必须要植入不同语言sdk到微服务中，通过接收不同语言中的agent 导出数据（opentrace、opentelemetry等协议），梳理不同agent的span上下级关系，形成网络拓扑图。

① 传统网络拓扑图-架构图

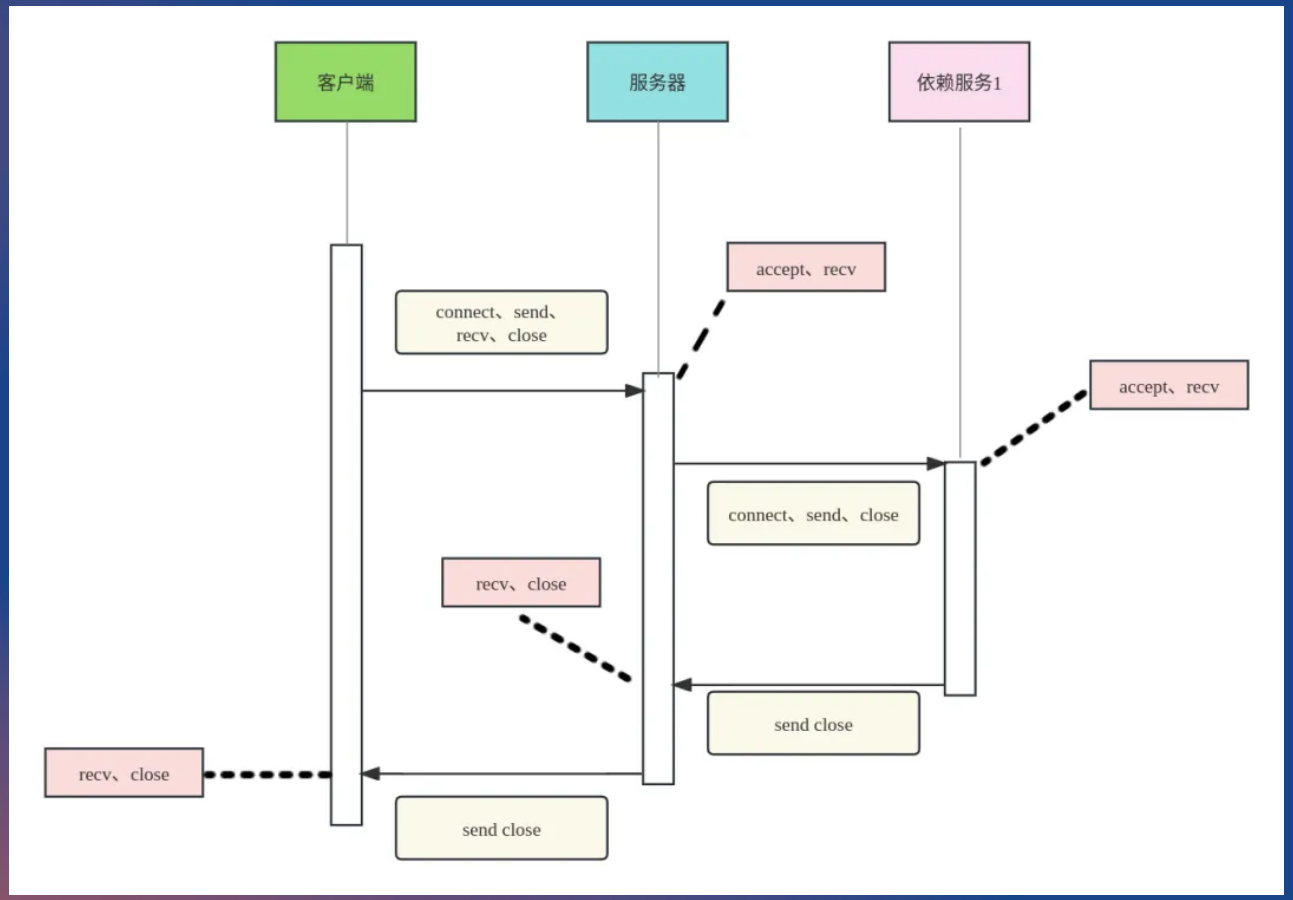


微服务Demo架构图

① 传统网络拓扑图-缺陷与不足

1、具有强侵入性，需要在服务中切入 不同语言的agent
2、维护成本高，需要支持不同类型的trace协议，并且自行维护多版本agent。
3、侵入agent，对服务性能有一定影响
4、拓扑只能反应服务间的调用关系，无法准确反映主机之间的网络拓扑全貌。

②基于ebpf网络拓扑图，我们首先要了解linux处理请求过程:



②ebpf可观测内核插桩常用技巧:

eBPF支持的内核探针功能，内核动态探针可以分为两种：Kprobe和Kretprobe。二者的区别在于，根据探针执行周期的不同阶段，来确定插入eBPF程序的位置。

kprobe

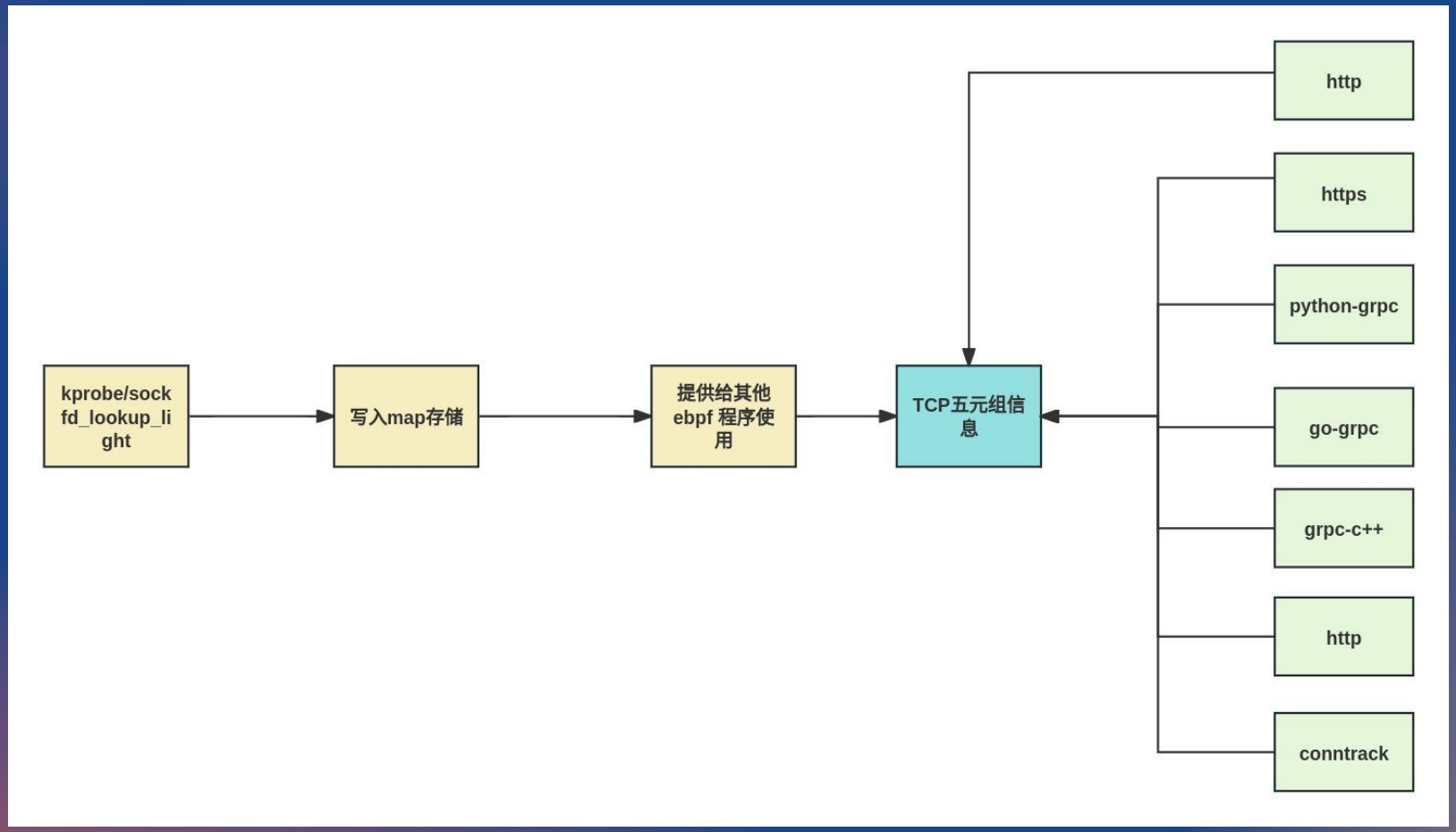
tracepoint是一个静态的hook函数，是预先在内核里面编写好才使用。使用tracepoint来监视网络设备队列的操作。

tracepoint

SOCKET FILTER 类型 eBPF程序通过 SO_ATTACH_BPF选项完成设置。SO_ATTACH_BPF插入的是eBPF代码。eBPF是对cBPF的增强，目前用户端的tcpdump等程序还是用的cBPF版本，他具有开销小，可编程等优点。

网络过滤器

使用kprobe获取pid 和fd 的对应关系:



sockfd_lookup_light 使用的系统调用有:

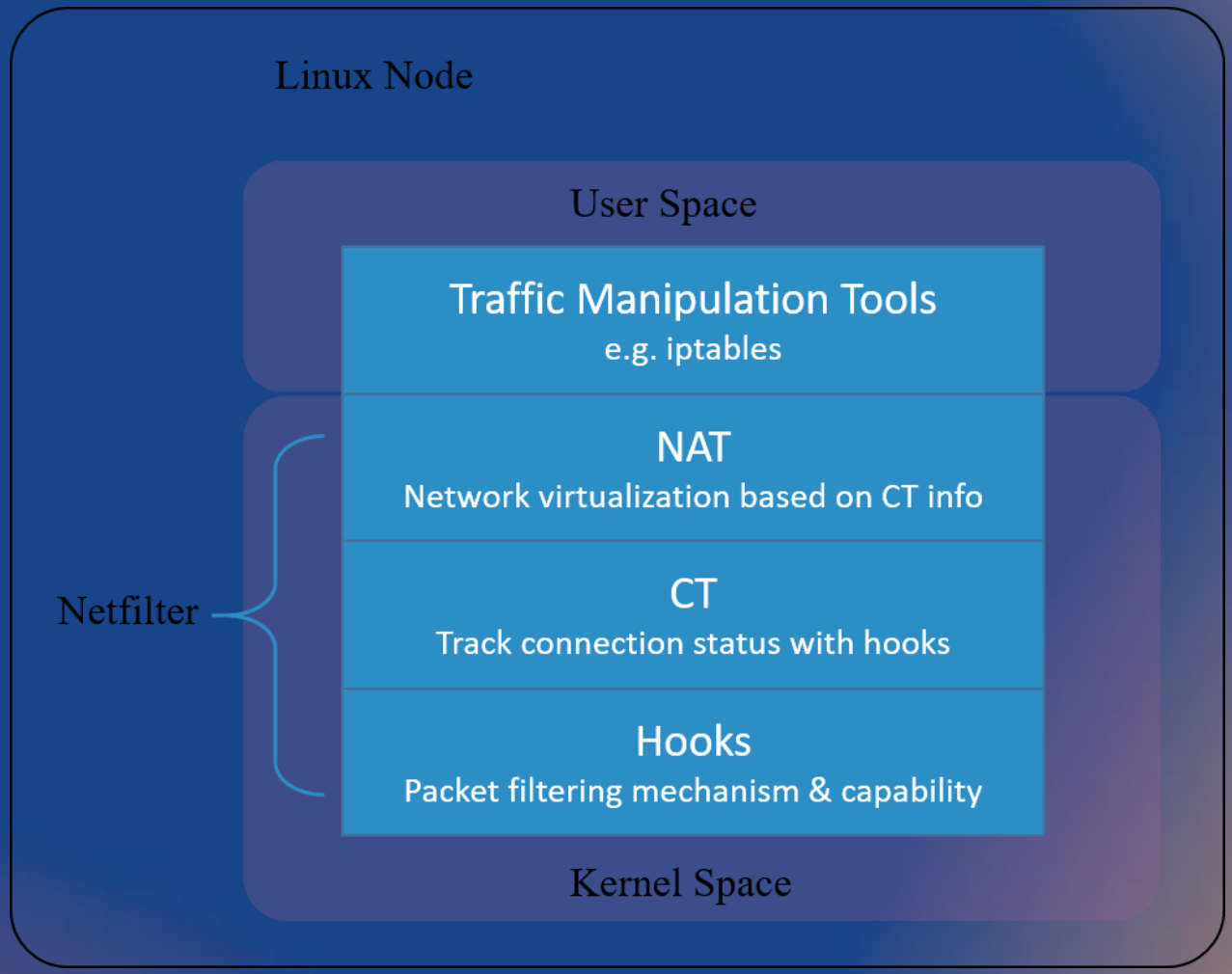
系统调用	是否使用sockfd_lookup_light
bind	是
listen	是
getsockname	是
getpeername	是
recvfrom	是
sendto	是
setsockopt	是
getsockopt	是
shutdown	是
sendmsg	是
recvmsg	是
send	是
recv	是
read	否
write	否
accept	否

标记流量方向:

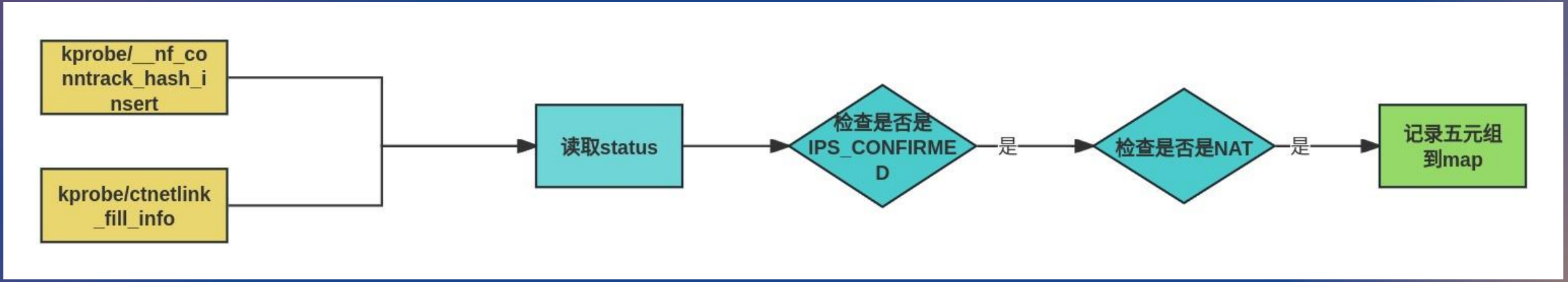
通过hook accept 标记处的入口流量
SEC("kretprobe/inet_csk_accept")

通过hook connect 标记处的出口流量
SEC("kprobe/tcp_connect")
SEC("kprobe/tcp_finish_connect")

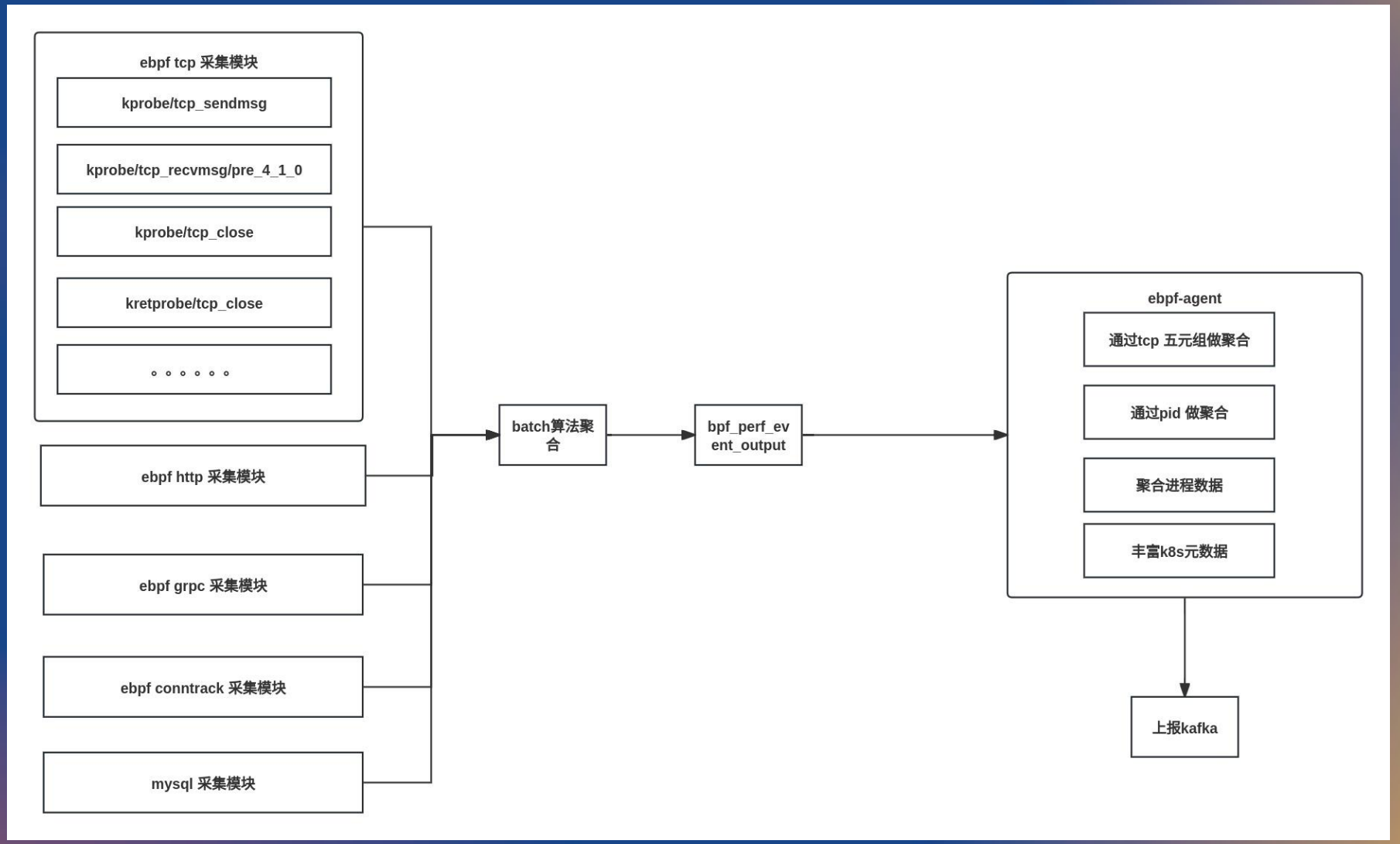
如何读取nat转换信息?



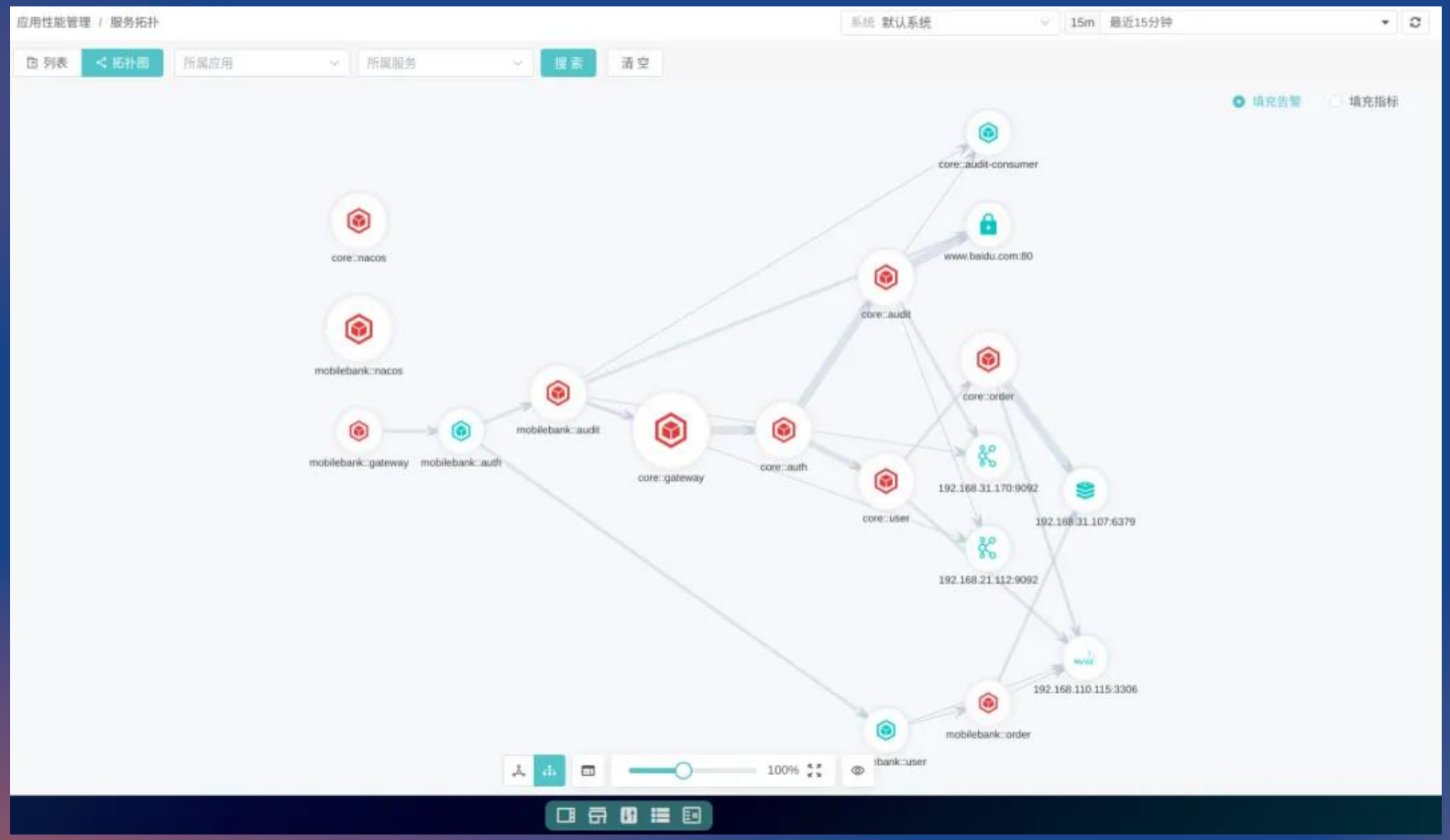
ebpf 插桩 netfilter



软件程序设计:


ebpf tcp 采集模块
kprobe/tcp_sendmsg
kprobe/tcp_recvmsg/pre_4_1_0
kprobe/tcp_close
kretprobe/tcp_close
。。。。。。
ebpf http 采集模块
ebpf grpc 采集模块
ebpf conntrack 采集模块
mysql 采集模块
batch算法聚合
bpf_perf_event_output
ebpf-agent
通过tcp 五元组做聚合
通过pid 做聚合
聚合进程数据
丰富k8s元数据
上报kafka

服务端汇聚网络拓扑图:


应用性能管理 / 服务拓扑
系统 默认系统
15m 最近15分钟
列表
拓扑图
所属应用
所属服务
搜索
清空
填充告警
填充指标
core::audit-consumer
core::nacos
www.baidu.com:80
core::audit
mobilebank::nacos
core::order
mobilebank::audit
mobilebank::gateway
mobilebank::auth
core::gateway
core::auth
192.168.31.170:9092
core::user
192.168.31.107:6379
192.168.21.112:9092
192.168.110.115:3306
mobilebank::order
100%

第二届 eBPF开发者大会

www.ebpftravel.com

二无侵入的流量metric统计

中国·西安

前期调研:

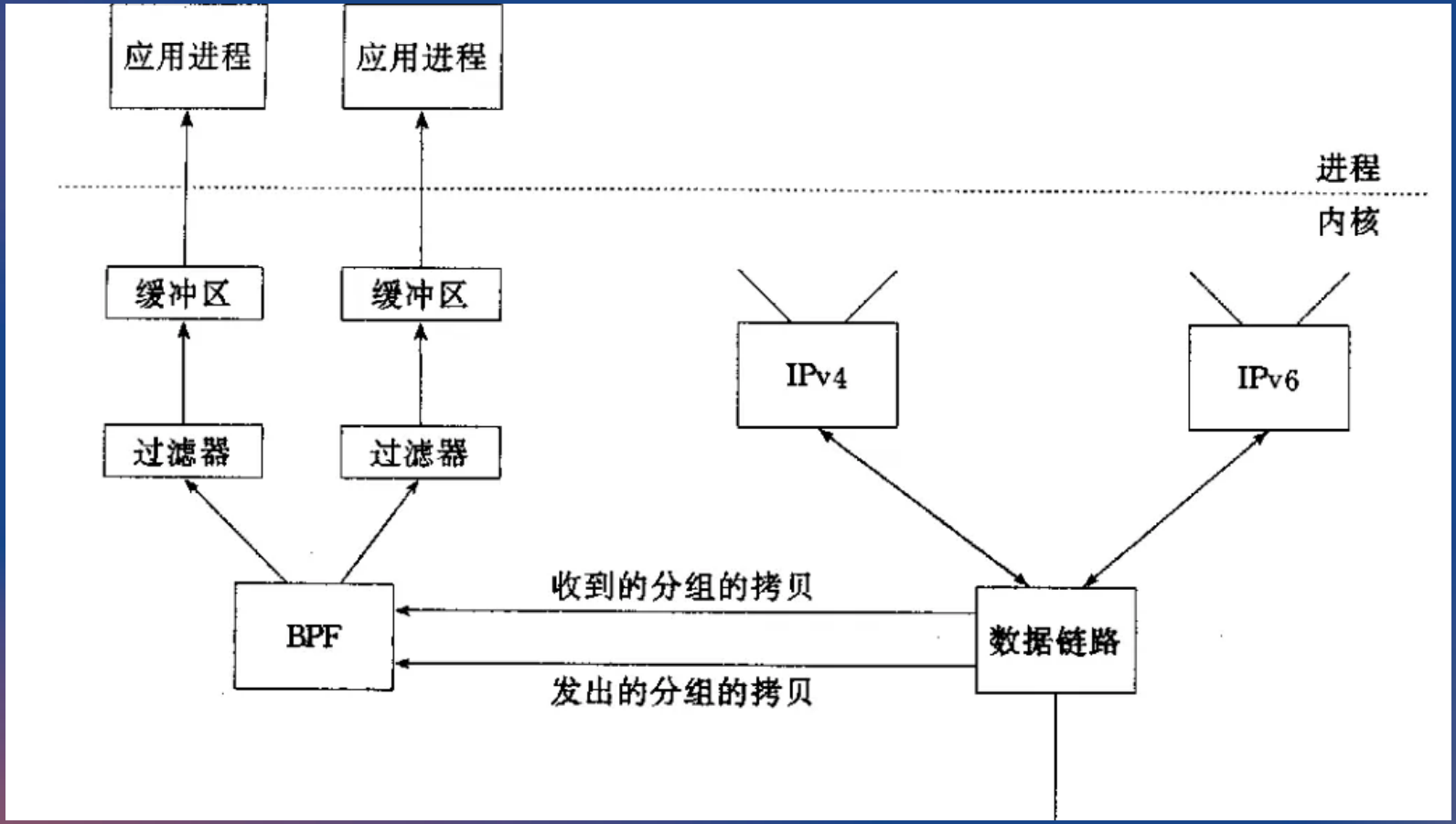
协议、微服务框架及eBPF采集支持状态

使用热度	协议	Golang框架	Java框架	协议描述	eBPF支持状态
1	HTTP	Dubbo-go、go-micro、go-zero、kratos、CloudWeGo-Kitex、Goa、Jupiter、tars-go	SpringBoot, Dubbo	https://cn.dubbo.apache.org/zh/docs3-v2/java-sdk/reference-manual/protocol	支持，hook内核函数
2	HTTPS	Dubbo-go、go-micro、go-zero、kratos、CloudWeGo-Kitex、Goa、Jupiter、tars-go	SpringBoot, Dubbo	https://cn.dubbo.apache.org/zh/docs3-v2/java-sdk/reference-manual/protocol	Go/Python支持，hook ssl库函数； Java不支持
3	gRPC	Dubbo-go、go-micro、CloudWeGo-Kitex、Jupiter、go-zero、kratos、Goa	SpringBoot, Dubbo	https://grpc.io/ 基于HTTP 2	Go/Python支持，hook grpc库函数 Java不支持
4	Thrift	Dubbo-go、CloudWeGo-Kitex	Dubbo、SpringBoot	https://thrift.apache.org/ 基于TCP	暂不支持
5	Dubbo	Dubbo-go	Dubbo	https://cn.dubbo.apache.org/zh/docs3-v2/java-sdk/reference-manual/protocol	暂不支持
6	Triple	Dubbo-go	Dubbo	https://cn.dubbo.apache.org/zh/docs3-v2/java-sdk/concepts-and-architecture/triple/	暂不支持
7	Webservice		Dubbo、SpringBoot	https://cn.dubbo.apache.org/zh/docs3-v2/java-sdk/reference-manual/protocol/webservice/	暂不支持
8	Hessian	Dubbo-go	Dubbo、SpringBoot	https://cn.dubbo.apache.org/zh/docs3-v2/java-sdk/reference-manual/protocol/hessian/	暂不支持
9	RMI		Dubbo、SpringBoot	https://cn.dubbo.apache.org/zh/docs3-v2/java-sdk/reference-manual/protocol/rmi/	暂不支持
10	XDS	Dubbo-go	Dubbo	https://www.envoyproxy.io/docs/envoy/latest/api-	暂不支持

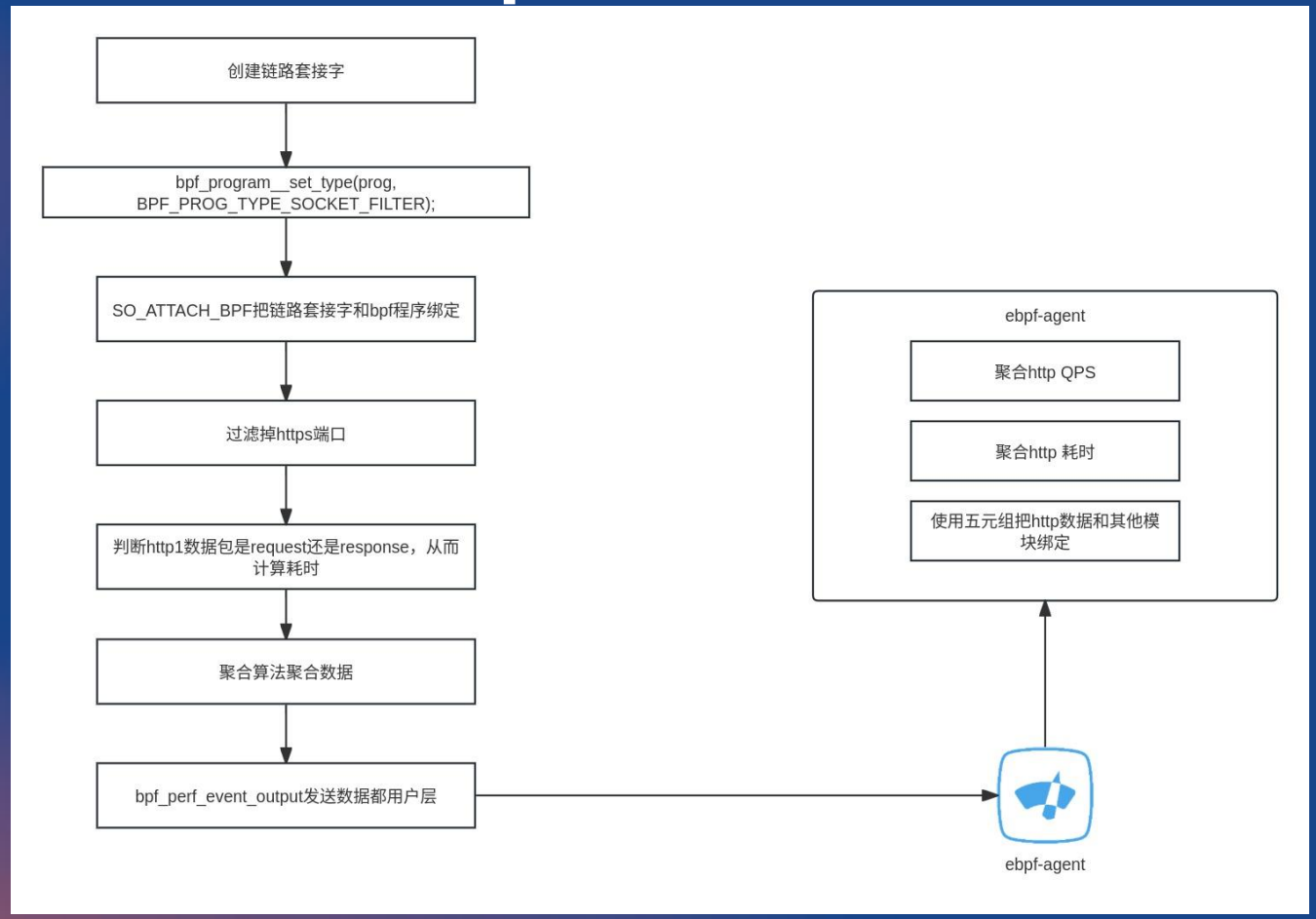
① 无侵入采集http指标

传统的http指标统计必须强植入代码，而且各个框架的底层函数不一样，开发维护成本都比较高，有很多时候我们需要为各个框架开发插件去进行上报统计，使用ebpf无侵入统计http指标解决了这个痛点。

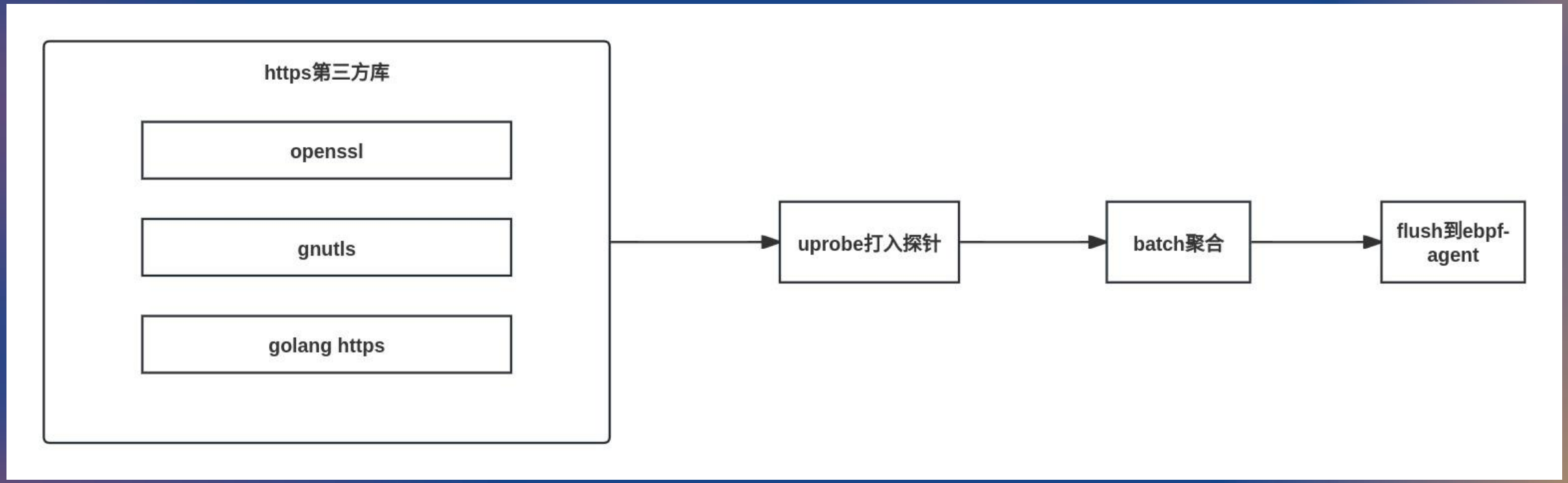
传统的CBPF统计有很大消耗



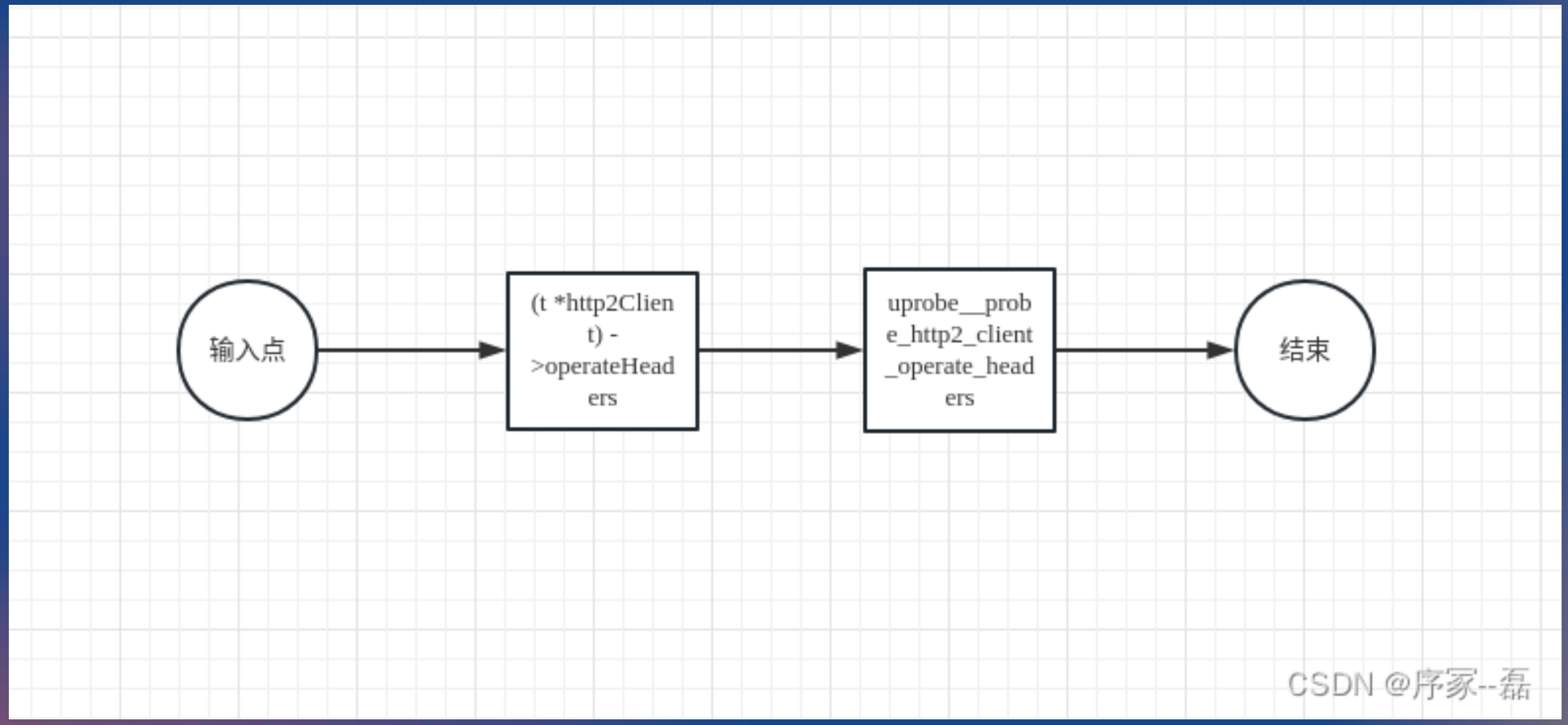
使用ebpf的低成本聚合http统计：



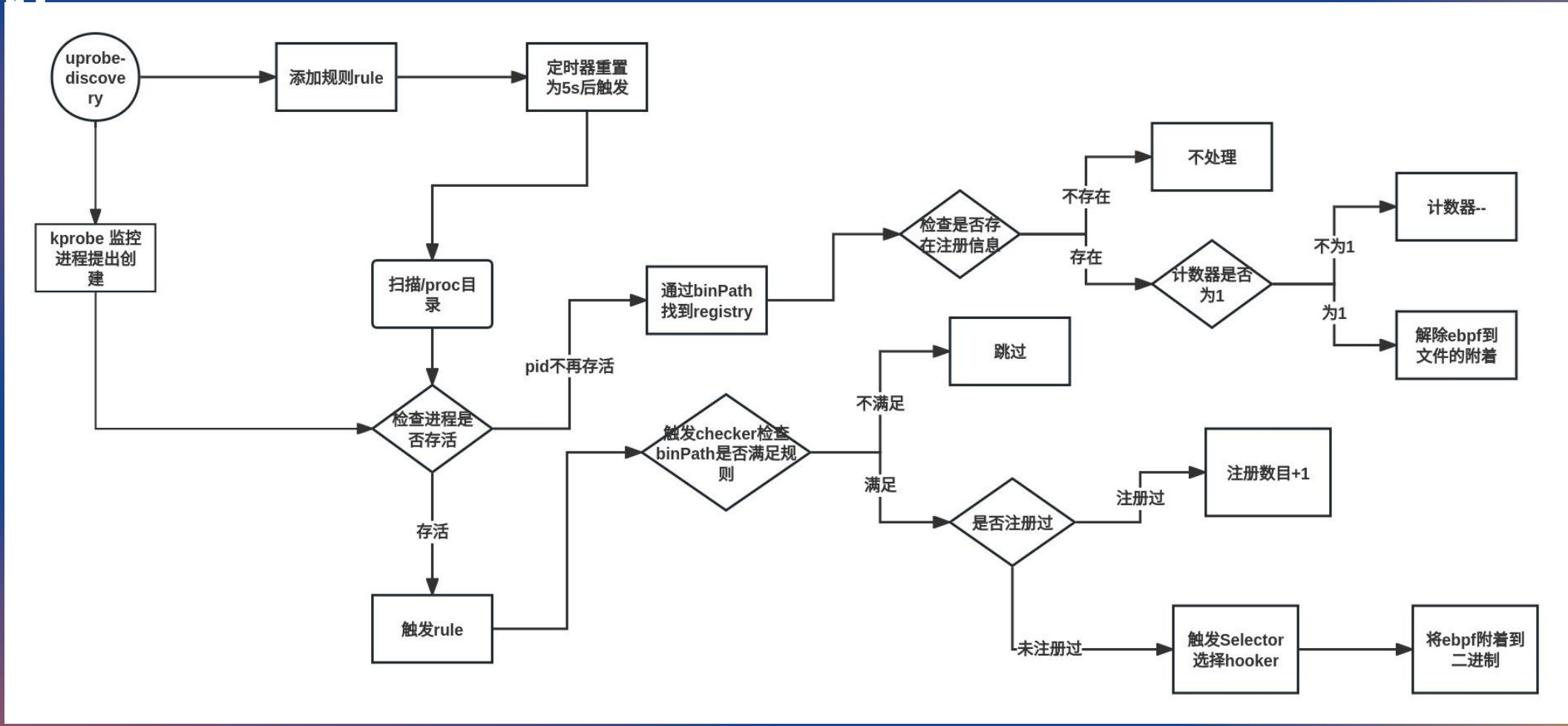
② uprobe采集https加密数据



③ uprobe采集grpc数据-uprobe植入hook点



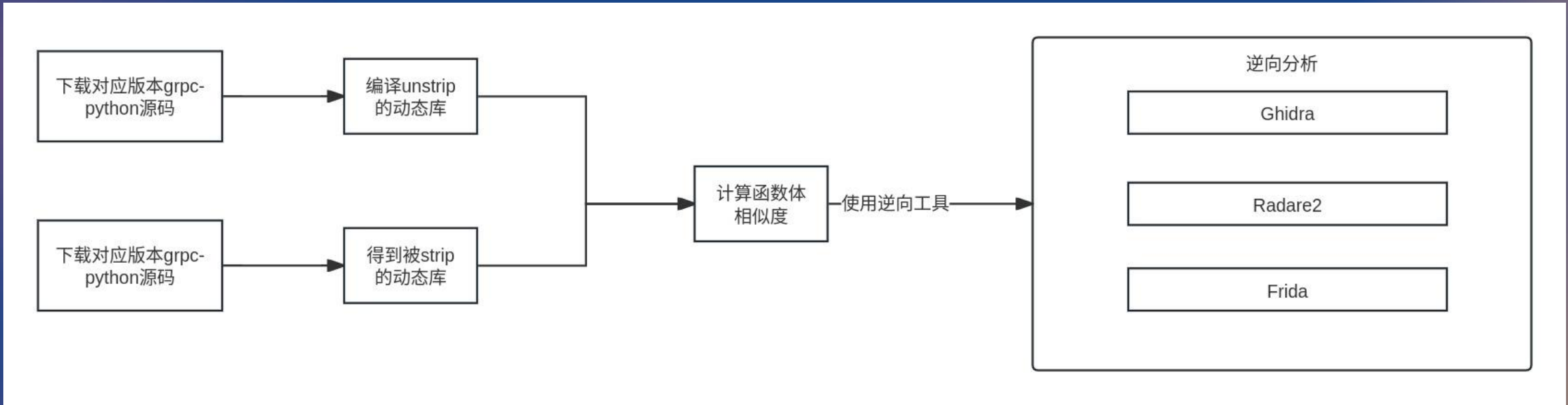
③ uprobe采集grpc-uprobe管理器设计



③ uprobe采集grpc-python库没有符号表怎么办?

每个python版本、每个grpc版本对应于不同的whl文件，以unzip解压后的so文件尺寸、build-id也都不相同。基于grpc源码按照官方流程基于docker构建的whl与pypi上同样版本的whl，其包含的so文件尺寸、build-id也都不相同。因此我们不得不逆向工程。

③ uprobe采集grpc-python库没有符号表怎么办?



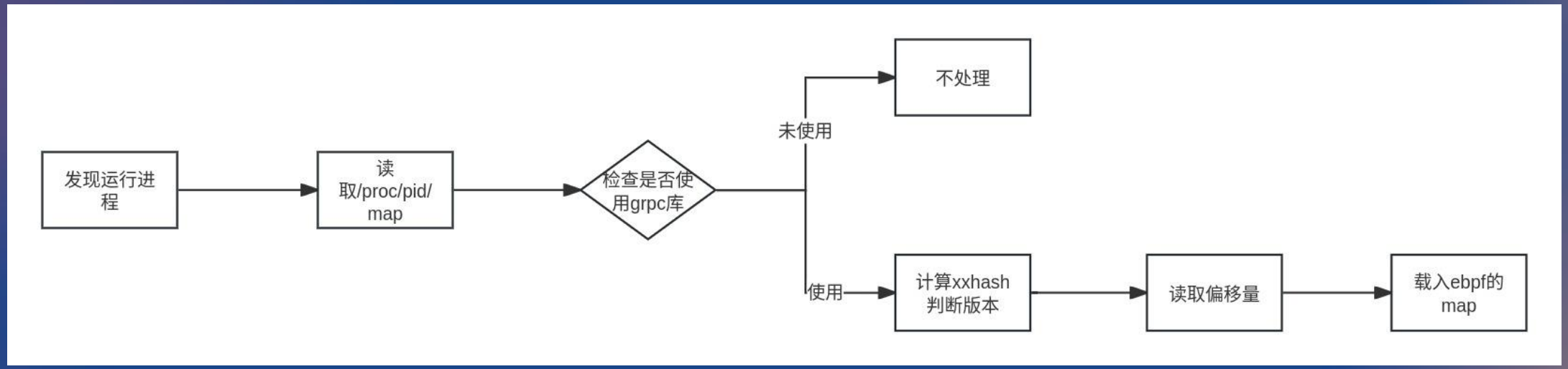
③ uprobe采集grpc-python库没有符号表怎么办?

1、通过逆向我们计算出要uprobe的函数的offset。

2、通过python-grpc.so文件二进制的xxhash值，识别出版本信息

3、通过ebpf系统调用把ebpf程序，植入python进程内。

③ uprobe采集grpc-python-拿到偏移量后怎么用



结果：

最终我们团队 成功逆向解析了200多个版本的python grpc 库 的关键插桩点的offset

第二届 eBPF开发者大会
www.ebpftravel.com

三 ebpf和可持续剖析

中国·西安

① 关于on-cpu和off-cpu

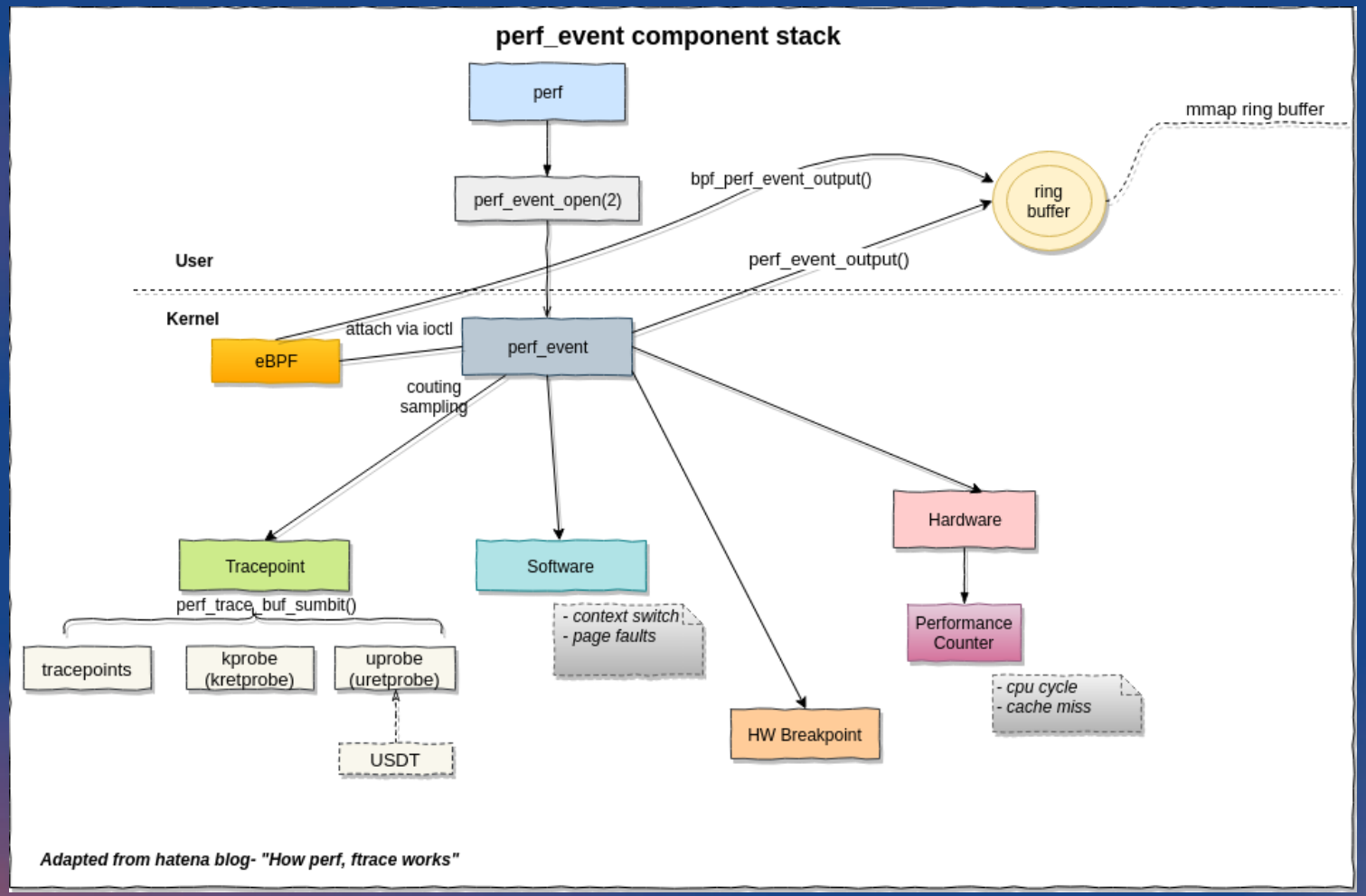
1、On-CPU 是指在 CPU 上执行的状态。

2、Off-CPU 是指不在 CPU 上运行的状态，包括在运行队列、阻塞睡眠等状态

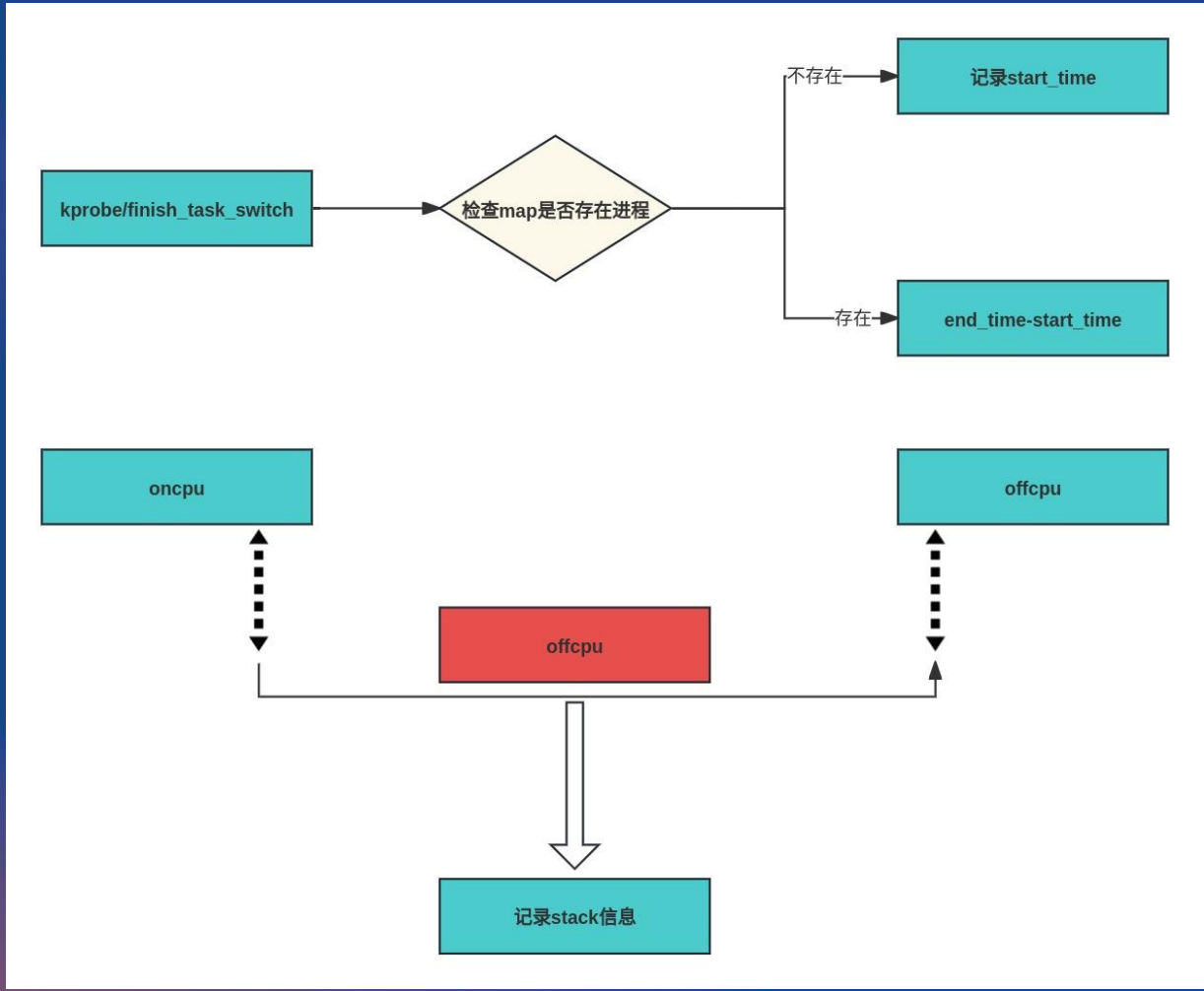
② hook点的选择finish_task_switch

finish_task_switch 是 Linux 内核中负责进行任务切换的一组函数。这个函数主要用于完成进程调度后的一些清理工作以及下一个要执行的进程的初始化工作。

③ 使用perf采集onCpu数据



④ 使用ebpf采集offcpu



四 ebpf 编程细节

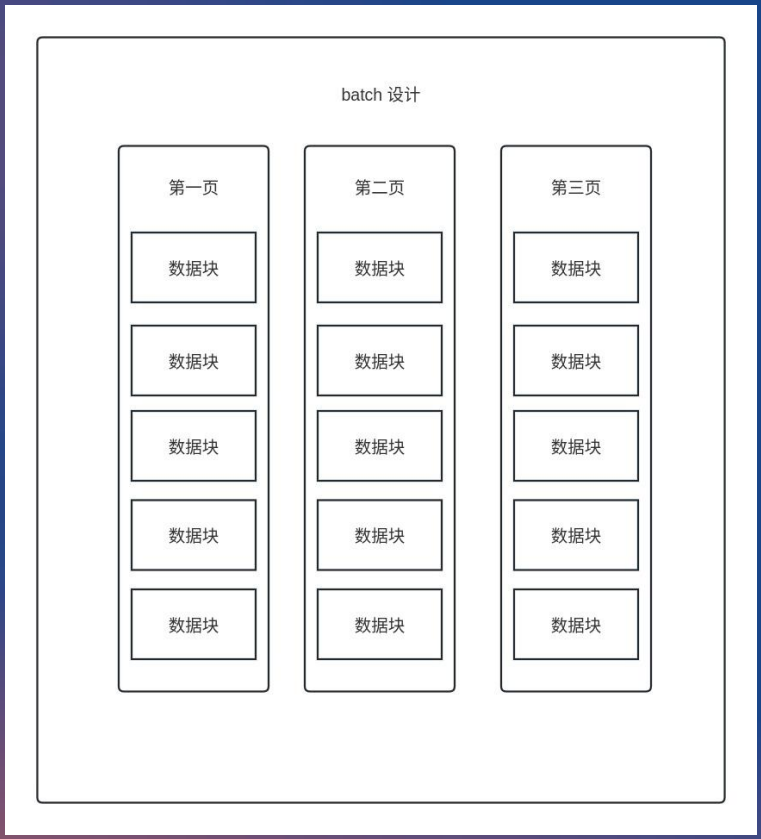
① ebpf申请大内存

使用BPF_MAP_TYPE_PERCPU_ARRAY,在ebpf 上申请大内存

```
__attribute__((section("maps"), used))
struct bpf_map_def tmp_storage_map = {
    .type = BPF_MAP_TYPE_PERCPU_ARRAY,
    .key_size = sizeof(u32),
    .value_size = PATH_MAX,
    .max_entries = 1,
};

res = bpf_probe_read(&pathname, sizeof(pathname), &regs->si)

map_id = 0;
map_value = bpf_map_lookup_elem(&tmp_storage_map, &map_id);
if (!map_value)
    return 0;

res = bpf_probe_read_str(map_value, PATH_MAX, pathname);
```

② 不要频繁使用bpf_perf_event_out

bpf_perf_event_out频繁触发会有性能影响，要做聚合



设计原则：

1、确保每个CPU一个batch

2、如果说所有的chunk满了，则停止写入

3、只有在一个chunk满了之后，才会使用bpf_perf_event_out导出数据到用户层

③ 谨慎使用for循环

ebpf 的默认栈大小最大 为512，所以写for循环一定要固定次数，越少越好

```
1
2
3  static __always_inline int for_test(lib_path_t *path, char *path_argument) {
4  #pragma unroll
5      for (int i = 0; i < 30; i++) {
6
7      }
8      return 0;
9  }
10
```

④ 关于如何计算offset-使用 gdb

复制

```
(gdb) ptype /o struct amba_device
/* offset      |    size */  type = struct amba_device {
/*      0      |     456 */    struct device {
/*      0      |      36 */        struct kobject {
/*      0      |       4 */            const char *name;
/*      4      |       8 */            struct list_head {
/*      4      |       4 */                struct list_head *next;
/*      8      |       4 */                struct list_head *prev;

                                        /* total size (bytes):    8 */
                                    } entry;
/*     12      |       4 */            struct kobject *parent;
/*     16      |       4 */            struct kset *kset;
/*     20      |       4 */            struct kobj_type *ktype;
/*     24      |       4 */            struct kernfs_node *sd;
/*     28      |       4 */            struct kref {
/*     28      |       4 */                refcount_t refcount;

                                        /* total size (bytes):    4 */
                                    } kref;
/*     32: 0   |       4 */            unsigned int state_initialized : 1;
/*     32: 1   |       4 */            unsigned int state_in_sysfs : 1;
/*     32: 2   |       4 */            unsigned int state_add_uevent_sent : 1;
/*     32: 3   |       4 */            unsigned int state_remove_uevent_sent : 1;
/*     32: 4   |       4 */            unsigned int uevent_suppress : 1;
/* XXX  3-bit padding    */
/* XXX  3-byte padding   */

                                    /* total size (bytes):   36 */
                                } kobj;
```

⑤ 关于如何计算uprobe offset的一些心得

如果遇到 带有 #define 的结构体，可能会导致release版本结构体offset发生变化，我们可以更改源码，编译成release后把他打印出来偏移量

```
std::cout << "read_closed offset:" <<
offsetof(grpc_chttp2_stream, read_closed) << std::endl;
```

第二届 eBPF开发者大会

www.ebpftravel.com

行业龙头共同的选择

中国·西安

第二届 eBPF开发者大会

www.ebpftravel.com

中国·西安

第二届 eBPF开发者大会

www.ebpftravel.com

clickhouse开源管理工具ckman

大模型知识库

第二届 eBPF开发者大会

www.ebpftravel.com

道 阻 且 长

感 谢 聆 听

中国·西安