哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

BoesFS

—基于eBPF实现的文件系统沙盒及其应用

队伍名称：BoesFS

技术主讲：杨大荣

01 FaaS发展

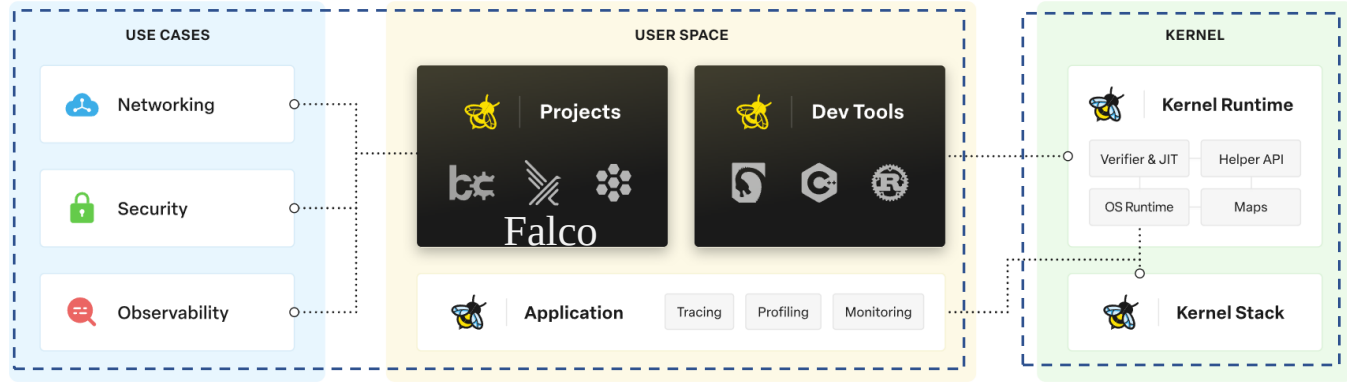
- **FaaS普及：**新一代无服务架构模式FaaS的发展，**成千上万的函数被交付FaaS平台**
- **FaaS挑战：**FaaS的函数服务带来了一个挑战，如何**安全且高性能**的运行数以万计的**非受信函数。**

- 以国内FaaS服务商为例，FaaS的开发者总和已经超过了 **50 万**，**每日超过 100 亿次** FaaS调用
- 随着FaaS的普及，云安全问题越发突显。2022 年 9 月，主流FaaS服务商微软云Azure的服务器遭到大规模数据泄露，影响超**548,000客户**，造成**2.4TB数据**泄露!

02 安全方式局限

- **FaaS需求：**非特权使用、自定义、细粒度、动态性、灵活性、热插拔、高性能
- **安全现状：**传统的安全访问控制技术已经无法很好满足FaaS的上述需求

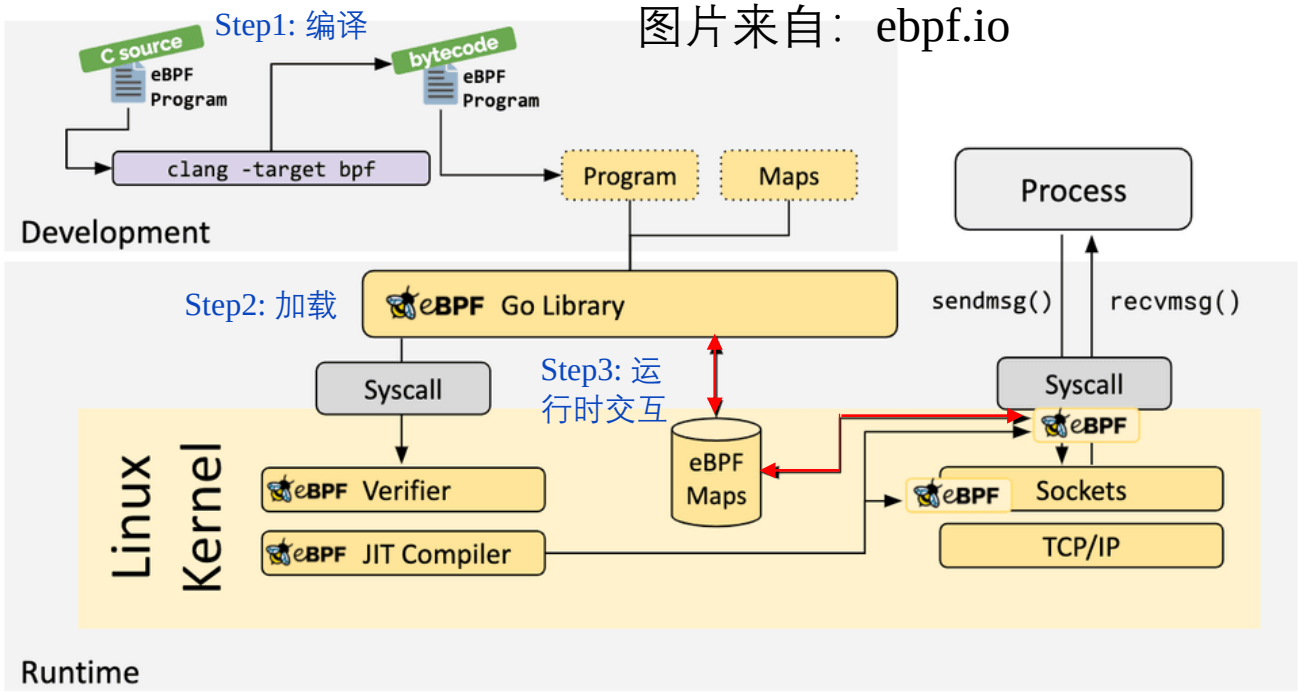
访问控制技术	所有用户	强安全	自定义	细粒度	动态性	灵活性	热插拔	性能开销
Namespace	×	√	×	×	×	×	×	–
DAC	√	√	×	×	√	×	×	–
MAC	×	√	√	√	√	√	×	–
Seccomp	√	√	√	√	×	×	×	–
LD_PRELOAD	√	×	√	√	√	×	√	低
PTRACE	√	√	√	×	×	×	×	≈50%
FUSE	√	×	√	√	×	×	×	≈80%
BoesFS	√	√	√	√	√	√	√	5%–7%



03 eBPF兴起

- **eBPF优势：**eBPF具有优秀的**运行时、内核空间和用户交互能力。**
- **eBPF现状：**eBPF性能、网络、资源、安全分析得到广泛应用。**但eBPF + 文件安全，缺少成熟的项目**做这方面的尝试

图片来自：ebpf.io



- 类虚拟机技术
- 用户空间与内核空间交互eBPF Map

01 FaaS发展

- **FaaS普及：**新一代无服务架构模式FaaS的发展，**成千上万的函数被交付FaaS平台**
- **FaaS挑战：**FaaS的函数服务带来了新挑战，如何**安全且高性能**的运行数以万计的**非受信函数。**

FaaS挑战

02 安全方式局限

- **FaaS需求：**非特权使用、自定义、细粒度、动态性、灵活性、热插拔、高性能
- **安全现状：传统的安全访问控制技术**已经**无法很好满足**FaaS上述需求。

安全现状

03 eBPF兴起

- **eBPF优势：**eBPF具有优秀的**运行时、内核空间和用户交互能力。**
- **eBPF现状：**eBPF性能、网络、资源、安全分析得到广泛应用。**但eBPF + 文件安全，缺少成熟的项目**做这方面的尝试

新兴工具

项目目标

目标	完成情况	完成细节
目标1：使用eBPF 技术实现文件系统沙盒	全部完成	① 结合eBPF技术和可堆叠文件系统，实现了**覆盖全部Linux VFS**的函数请求（20个API）的沙盒文件系统，BoesFS-in-Kernel。
目标2：提供细粒度的访问控制	全部完成	① 主体上，该文件系统沙盒能实现**进程级别**的细粒度访问控制。 ② 客体上，实现在**VFS最底层对象**(dentry等)级别的细粒度上对文件请求进行访问控制。
目标3：允许动态的自定义安全检查	全部完成	① 支持**三种自定义模式**：原生字节码、ACL模式、RBAC模式。（均支持参数匹配） ② 支持**完全动态性**。即使沙盒程序正在运行，也能进行安全检查规则的切换。
目标4：具备低性能开销	全部完成	① 基于**小而精**的eBPF字节码和**轻量级**可堆叠层，并基于perf瓶颈定位，进一步实现**ACL查询优化**。 ② 对BoesFS进行文件系统性能测试，BoesFS的性能开销仅在**5%-7%**。
目标5：提供安全且高性能的FaaS环境	全部完成	① 实现了以BoesFS沙盒为函数运行环境的**BoesFS-FaaS**，支持**自定义安全**检查规则。 ② 对BoesFS-FaaS进行并发测试，满足**高性能**的FaaS需求。

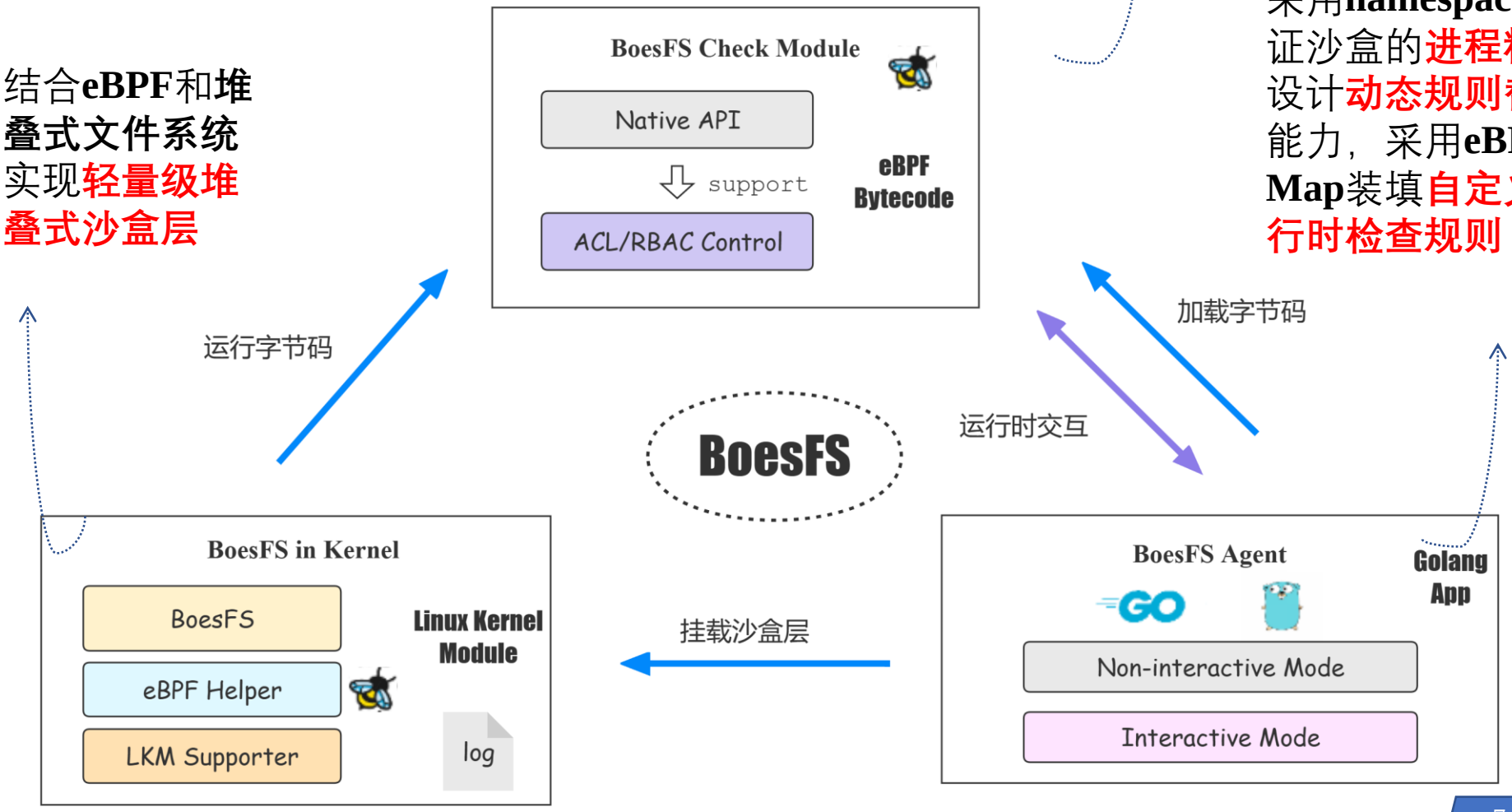
3.1 BoesFS 总体架构

构成组件示意

结合**eBPF**和**堆叠式文件系统**实现**轻量级堆叠式沙盒层**

使用**eBPF字节码**实现**ACL/RBAC访问控制模型**

采用**namespace**保证沙盒的**进程粒度**，设计**动态规则替换**能力，采用**eBPF Map**装填**自定义运行时检查规则**

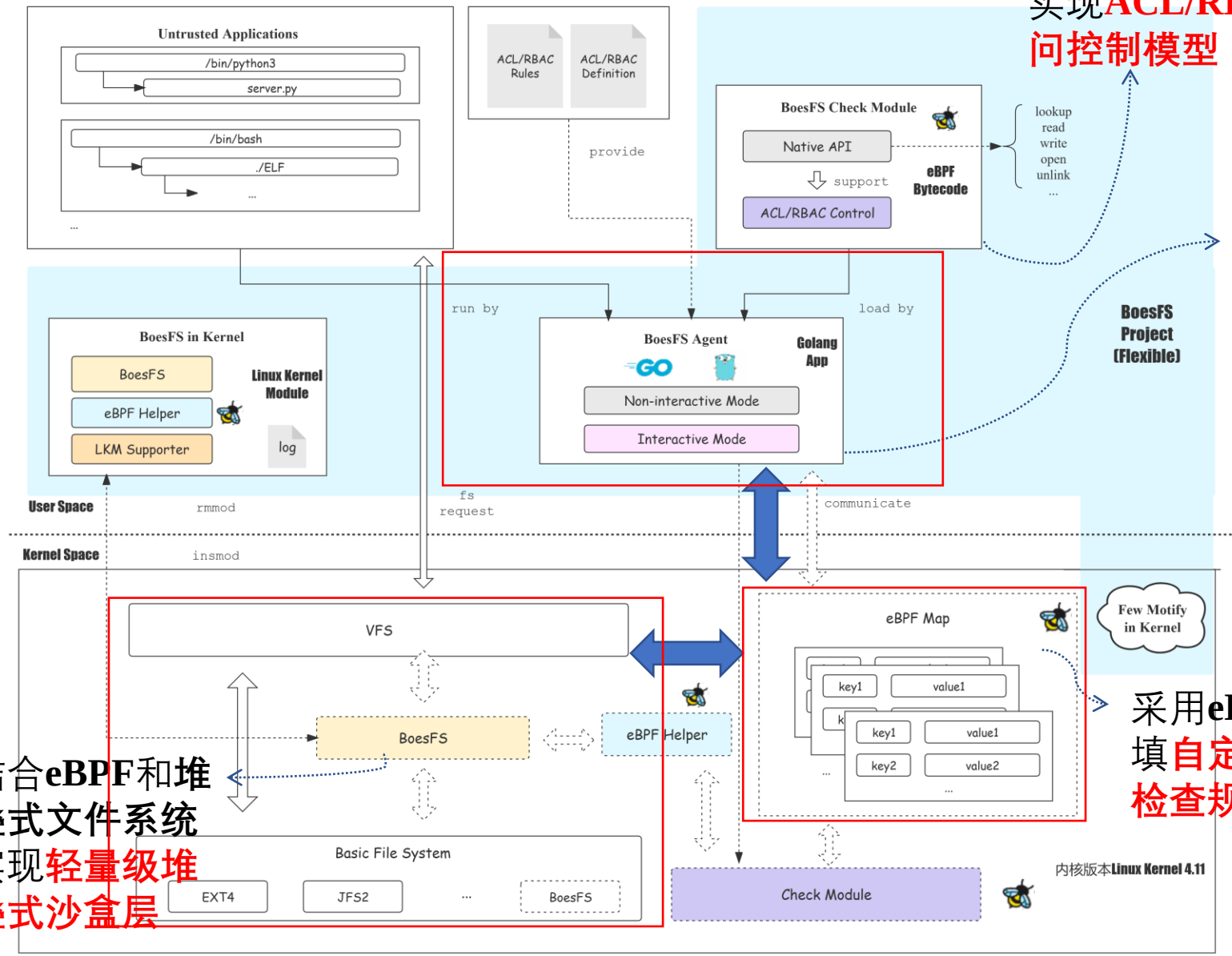


3.1 BoesFS 总体架构

使用**eBPF字节码**实现**ACL/RBAC访问控制模型**

采用**namespace**保证沙盒的**进程粒度**，设计**动态规则替换**能力

运行时结构示意



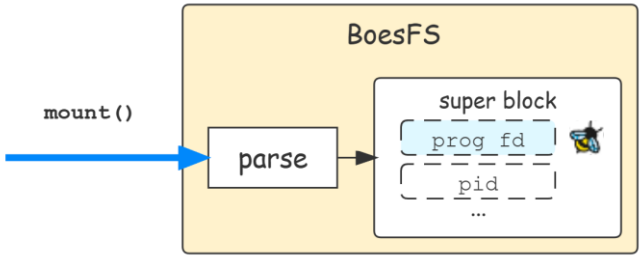
结合**eBPF**和**堆叠式文件系统**实现**轻量级堆叠式沙盒层**

采用**eBPF Map**装填**自定义运行时检查规则**

3.2 BoesFS Kernel 模块设计

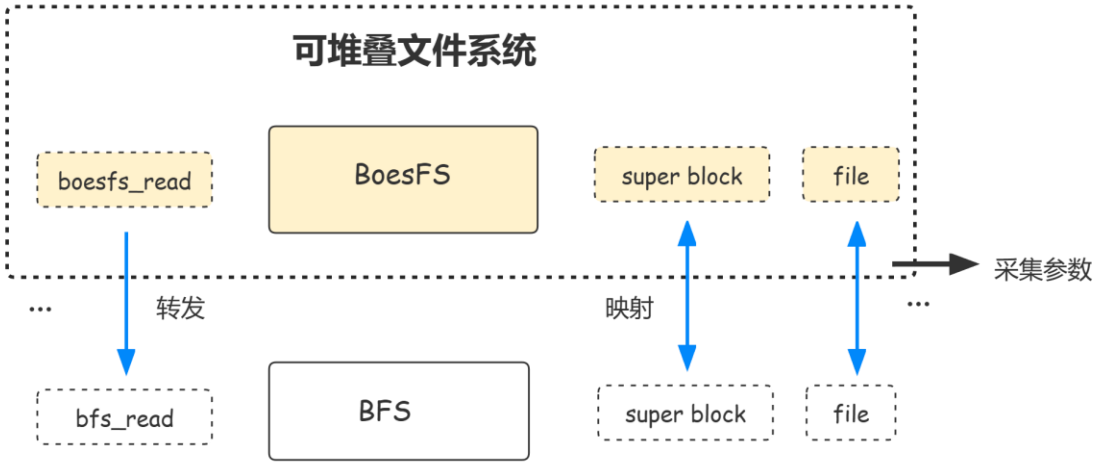
BoesFS 文件系统设计

沙盒状态维护：



- BoesFS Agent通过mount系统调用挂载一个BoesFS层，传递约定参数（如字节码装载fd）
- BoesFS文件系统对mount参数进行解析，解析的沙盒参数状态维护在文件系统的**超级块**中

参数采集：

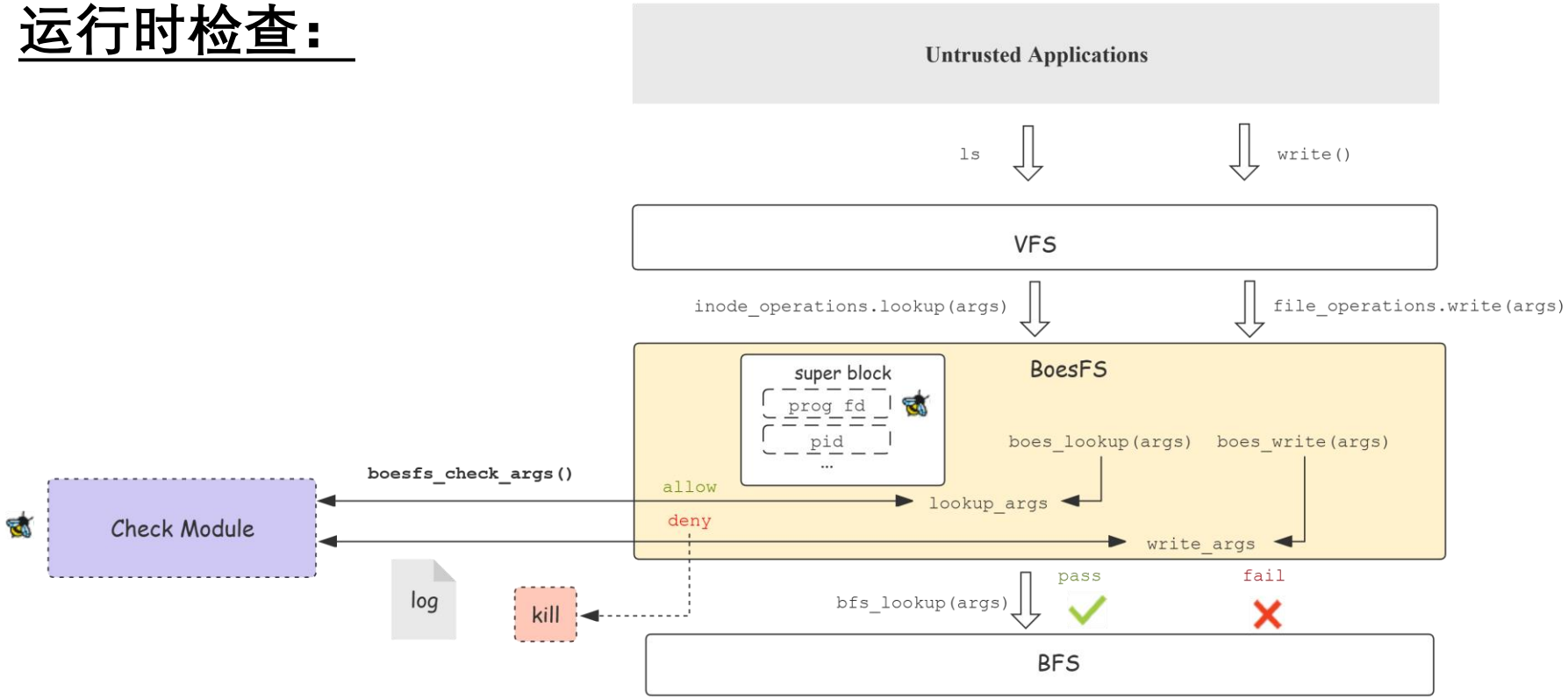


- 在BoesFS中，各个抽象对象都和底层BFS建立了**映射和转发**
- 我们基于此，完成每次发往BFS的文件请求的全部**参数采集**

3.2 BoesFS Kernel 模块设计

BoesFS 文件系统设计

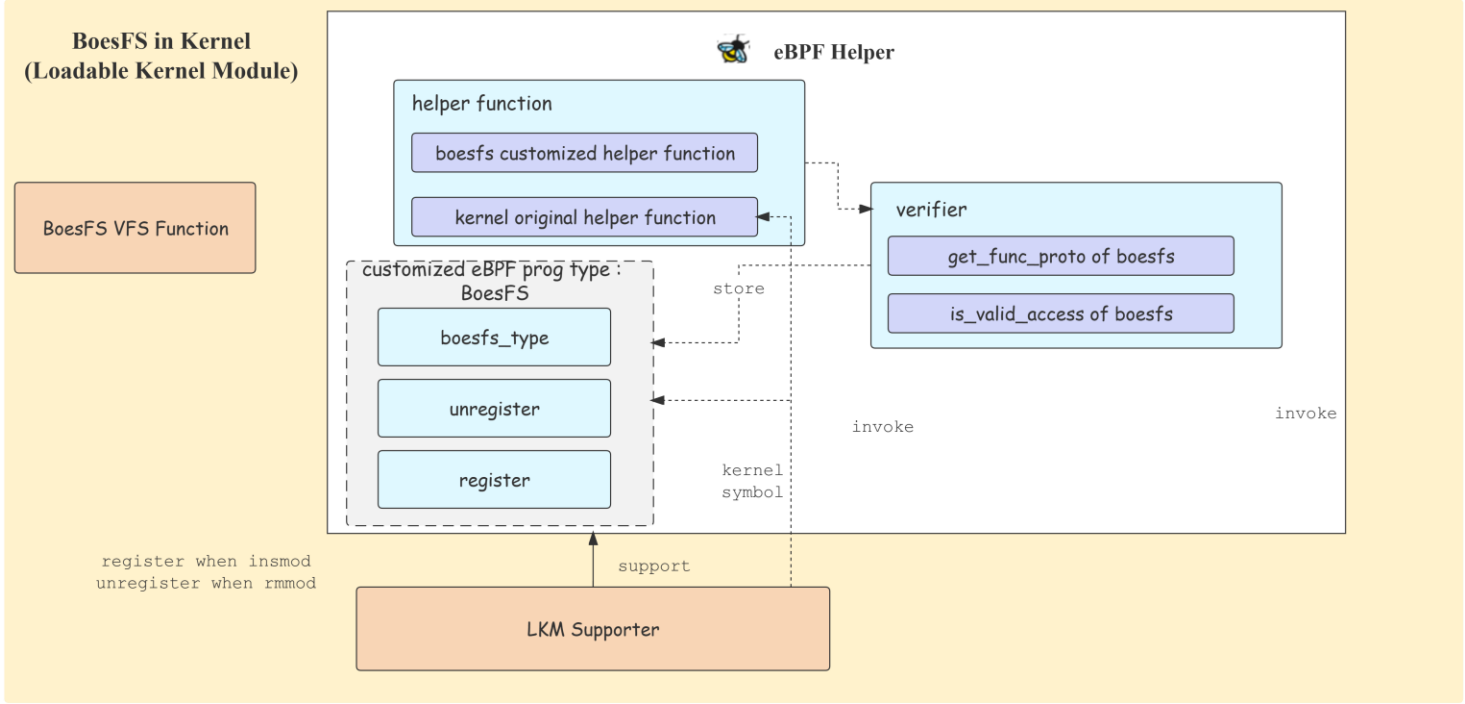
运行时检查：



- 所有的文件请求都会经过BoesFS层，介于VFS和底层文件系统（BFS）之间
- 收集到的参数会被传入到字节码检查模块
- 根据自定义的规则检查的结果进行放行和拦截

3.2 BoesFS Kernel 模块设计

eBPF支持的设计



- 定义**BoesFS特定的eBPF字节码类型**
- 实现字节码类型的注册和取消函数
- 实现eBPF字节码的验证器
- 实现BoesFS eBPF字节码专用工具函数

3.2 BoesFS Kernel 模块设计

其他功能的实现

文件请求审计：

- 收集到的参数的同时向用户态同步转发
- 内核线程，向用户态输出文件审计

终止异常进程：

- 在超级块维护沙盒进程的拒绝数
- 达到上限，从内核发送终止信号

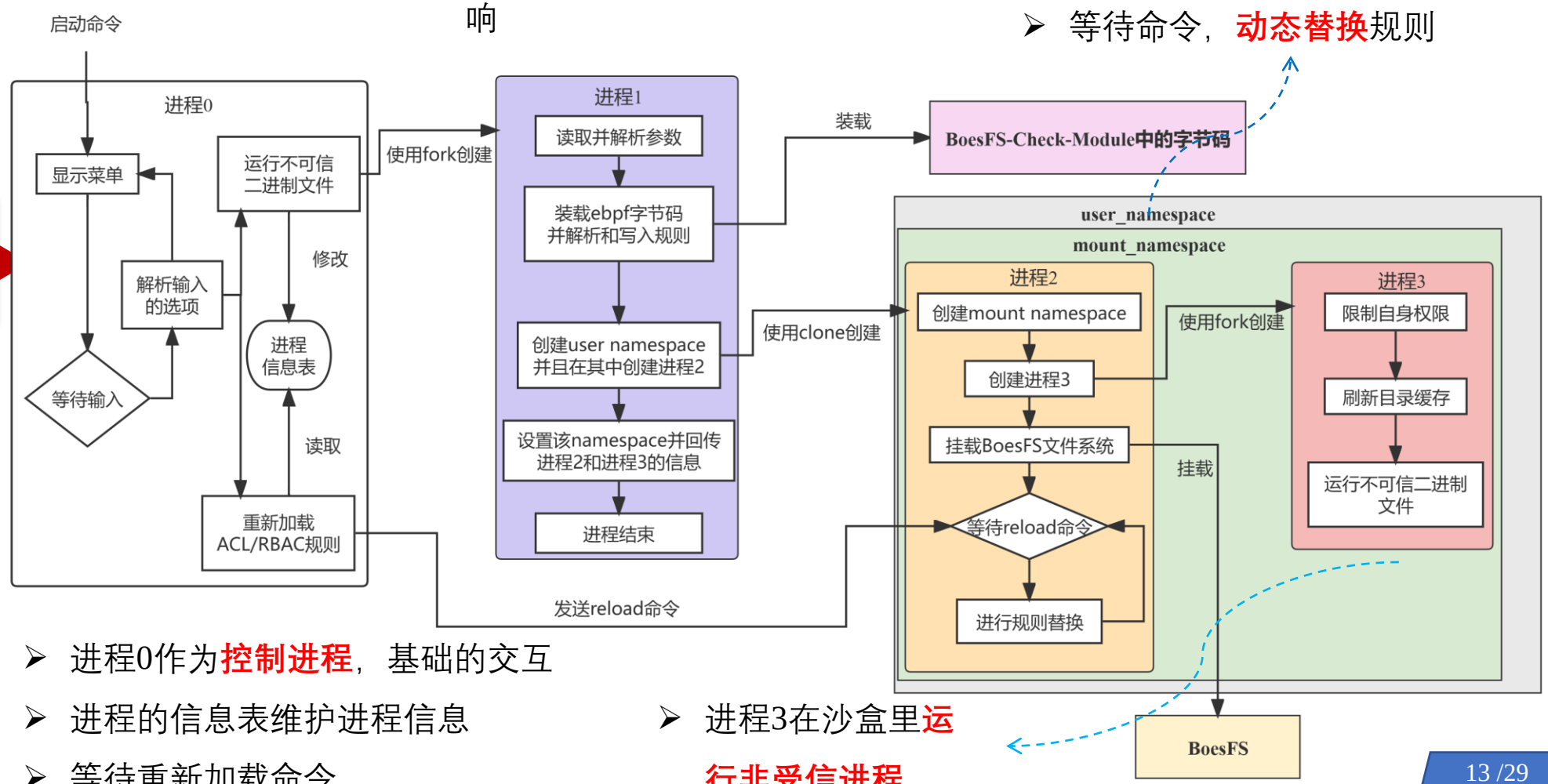
用户资源量配置：

- eBPF程序的需求，为用户分配使用memlock资源量
- 内核模块传参，修改rlimit配置文件

3.3 BoesFS Agent 模块设计

主要运行流程：

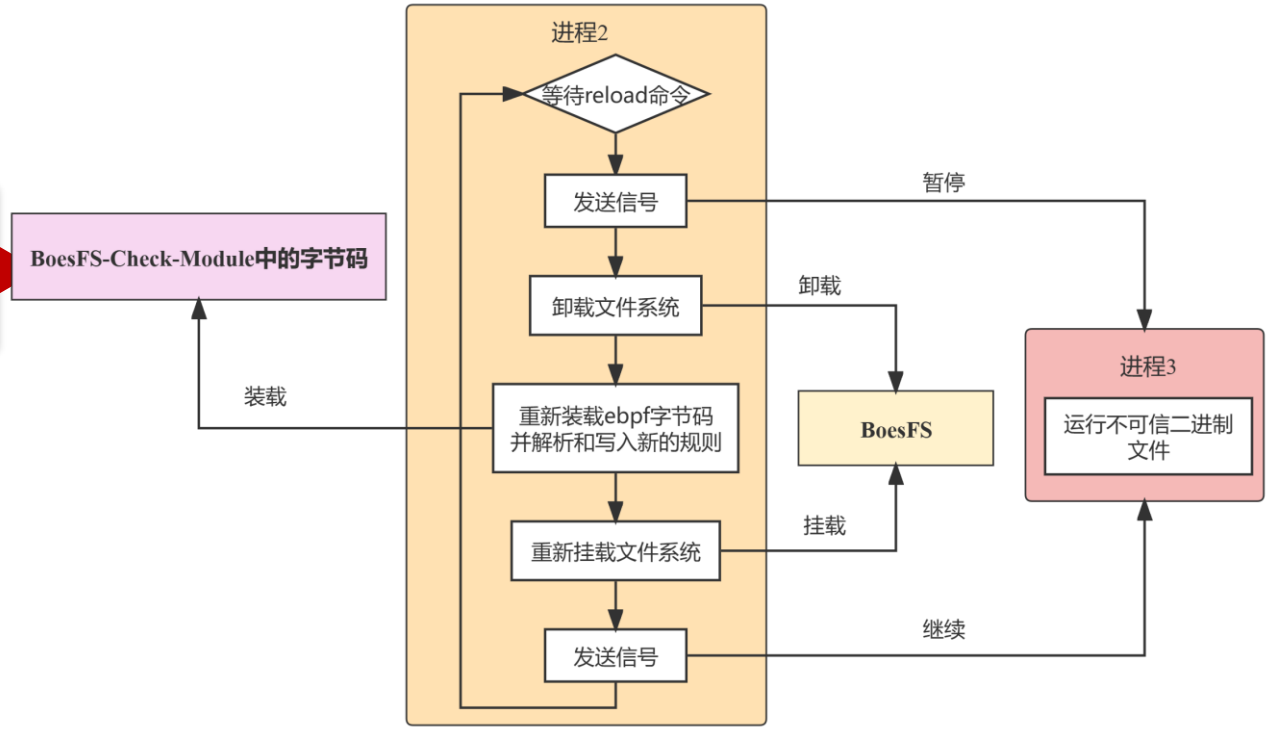
- 进程1装载**eBPF字节码**
- 填写用户规则到**eBPF Map**
- 创建**user namespace**，避免挂载影响
- 进程2处于独立的**mount namespace**
- **挂载沙盒层BoesFS**
- 等待命令，**动态替换**规则



- 进程0作为**控制进程**，基础的交互
- 进程的信息表维护进程信息
- 等待重新加载命令
- 进程3在沙盒里**运行非受信进程**

3.3 BoesFS Agent 模块设计

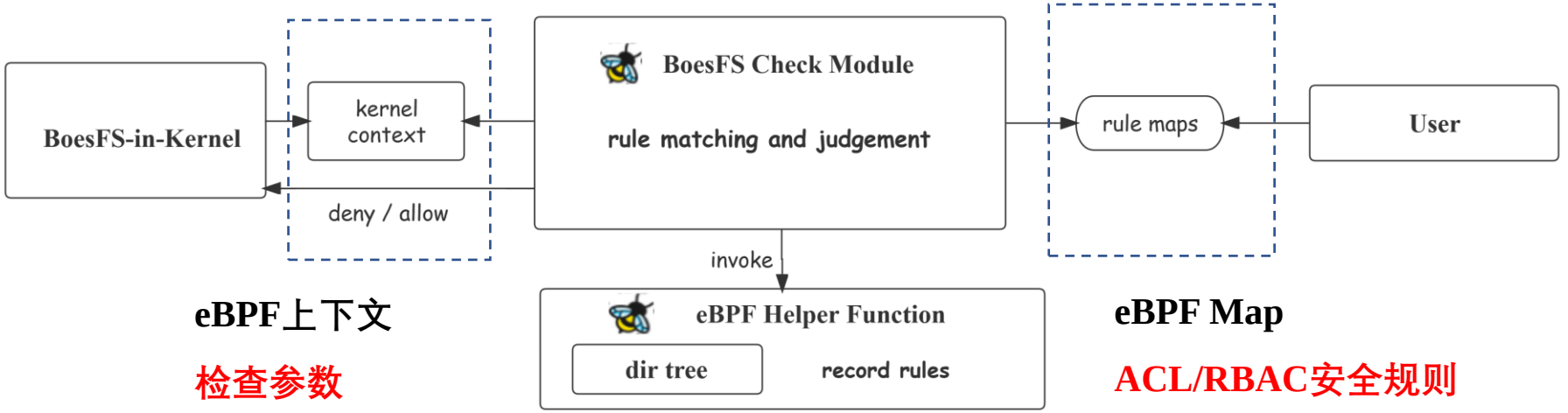
动态替换规则：



- 进程2负责动态替换规则
- **信号机制**，暂停和唤醒非受信进程的运行
- 卸载沙盒层，重新完成**规则载入**，**重新挂载**沙盒层

3.4 BoesFS Check Module 模块设计

两个交互通道： 实现**ACL/RBAC程序，参数匹配**的字节码



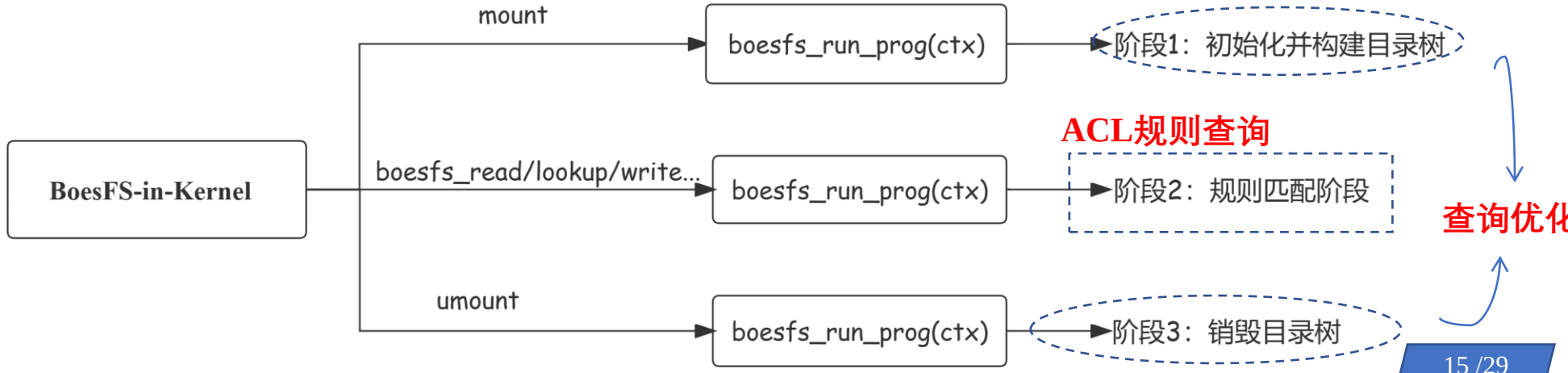
eBPF上下文

检查参数

eBPF Map

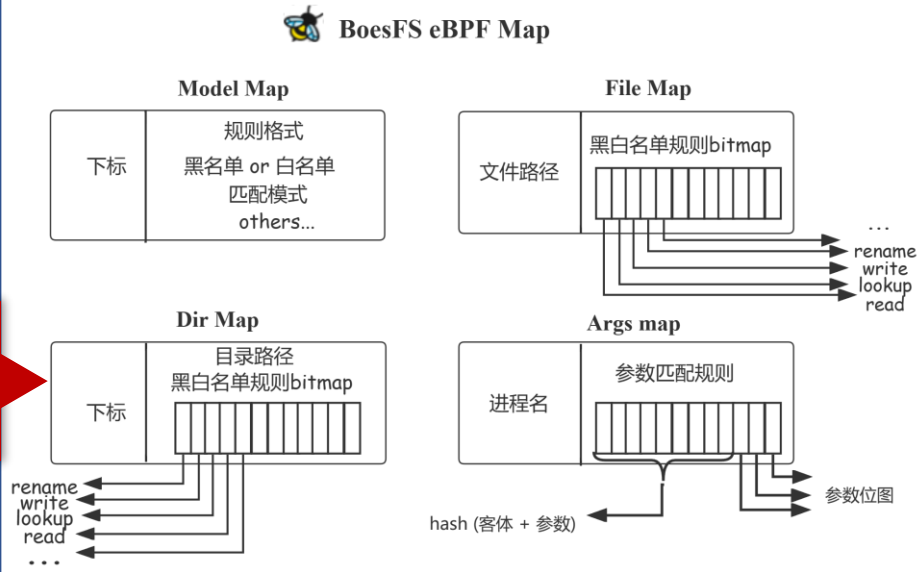
ACL/RBAC安全规则

三个运行阶段：



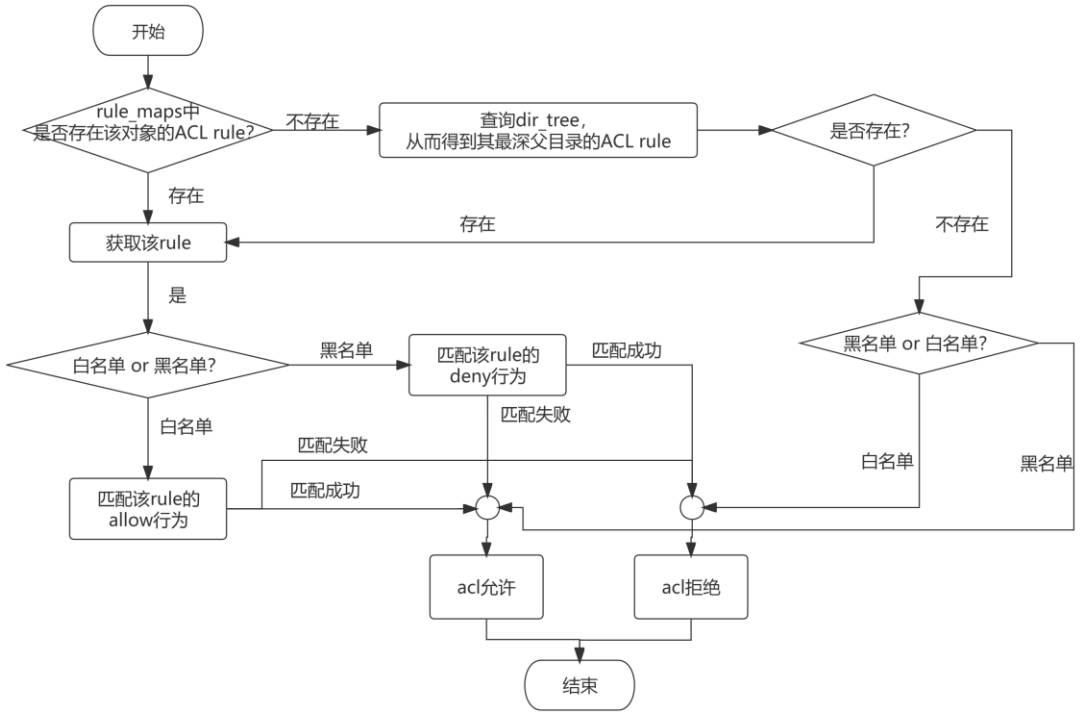
3.4 BoesFS Check Module 模块设计

BoesFS eBPF Map：



- 运行时交互，动态装载
- 存储**模型和规则**

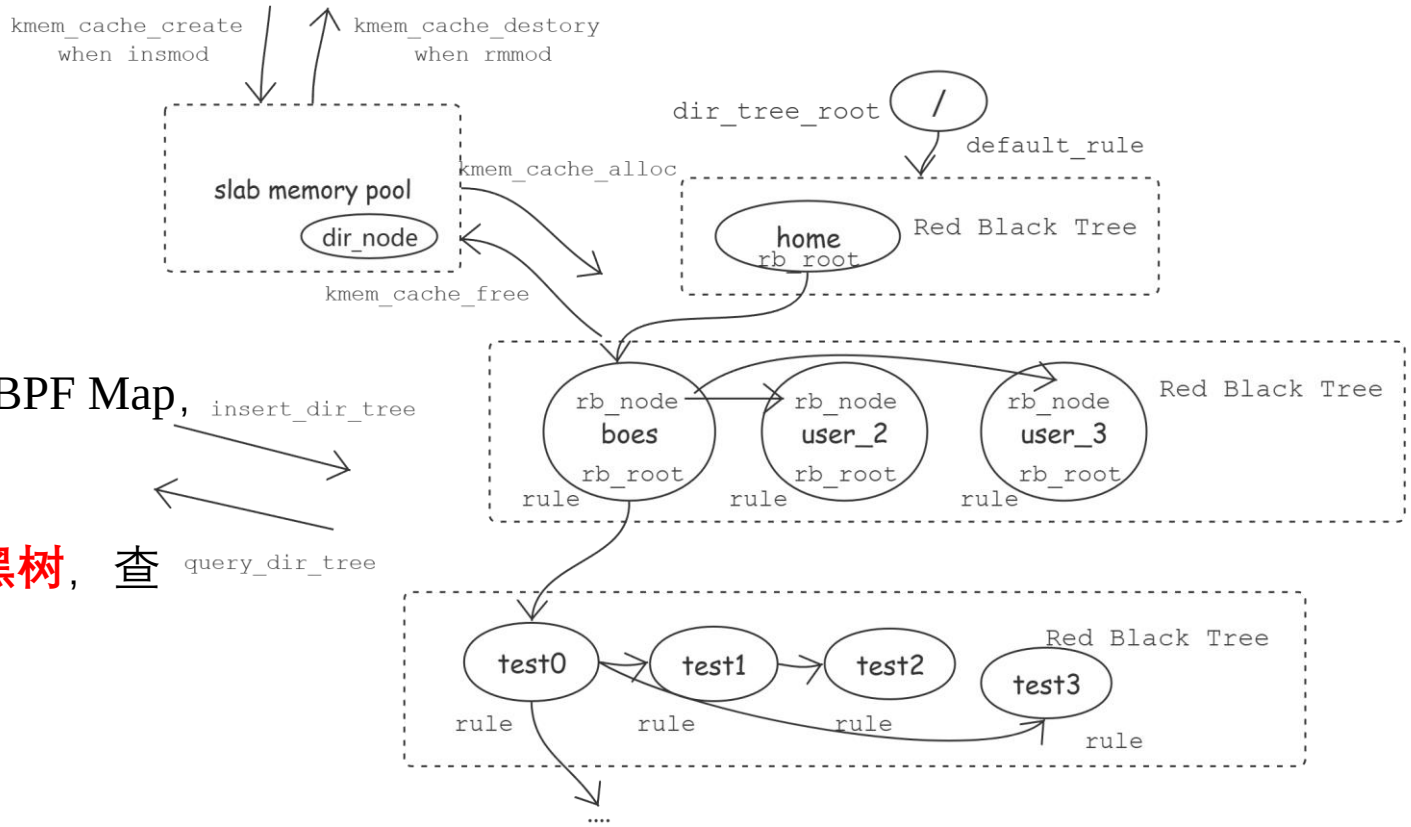
ACL规则查询：



- 查询**eBPF Map**，规则存储
- 最长路径匹配原则找**最深父目录**规则
- 提供**黑白名单**两种方式

3.4 BoesFS Check Module 模块设计

查询优化：

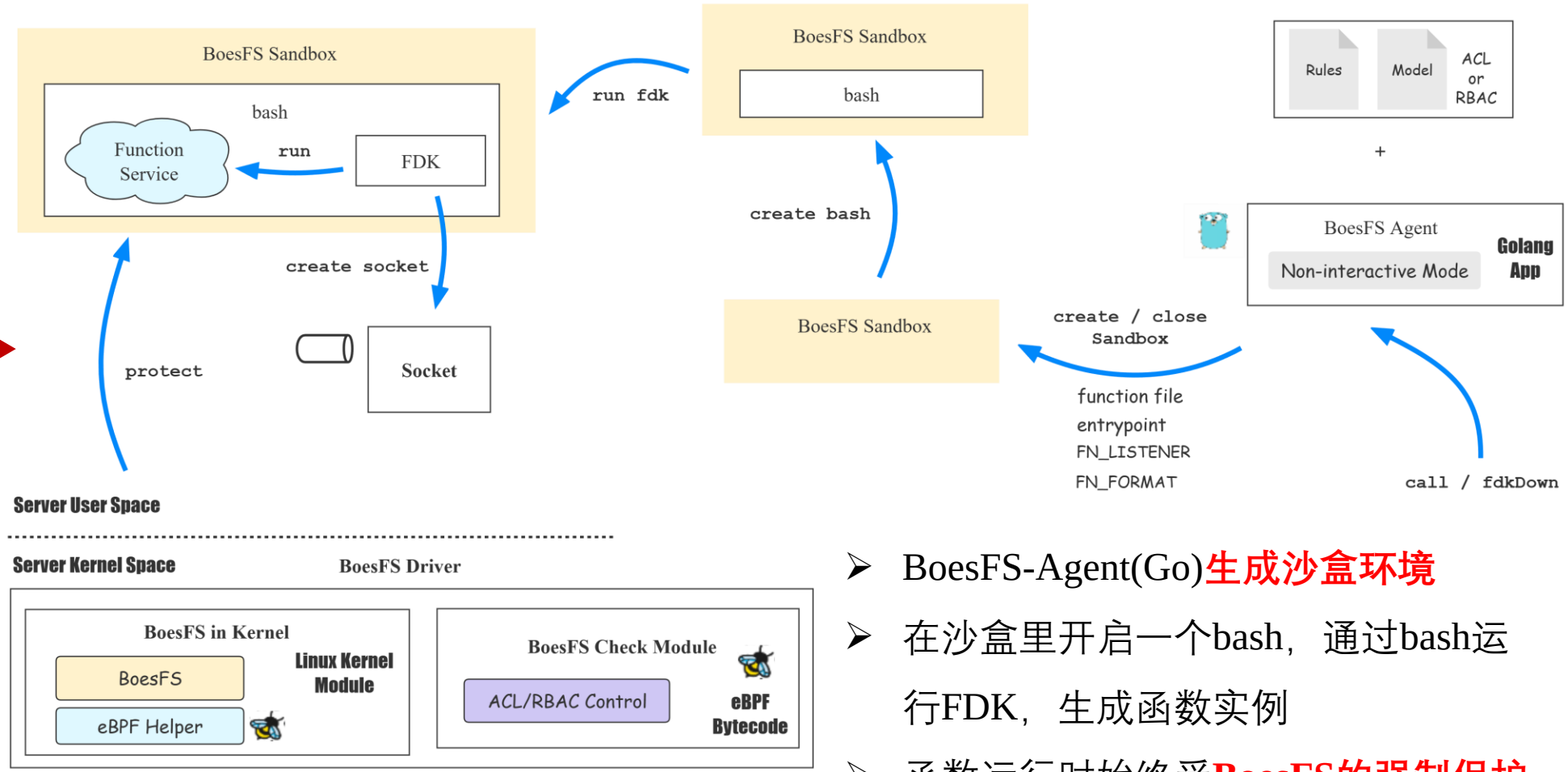


- 区域赛采用暴力遍历eBPF Map，查询效率低
- 决赛维护**字典树和红黑树**，查询速率大幅提高

指标	暴力遍历	查询优化	开销降低
查询操作开销 (ns/op)	1315.1	168.1	**87.2%**

3.5 BoesFS FaaS 设计

BoesFS沙盒环境：

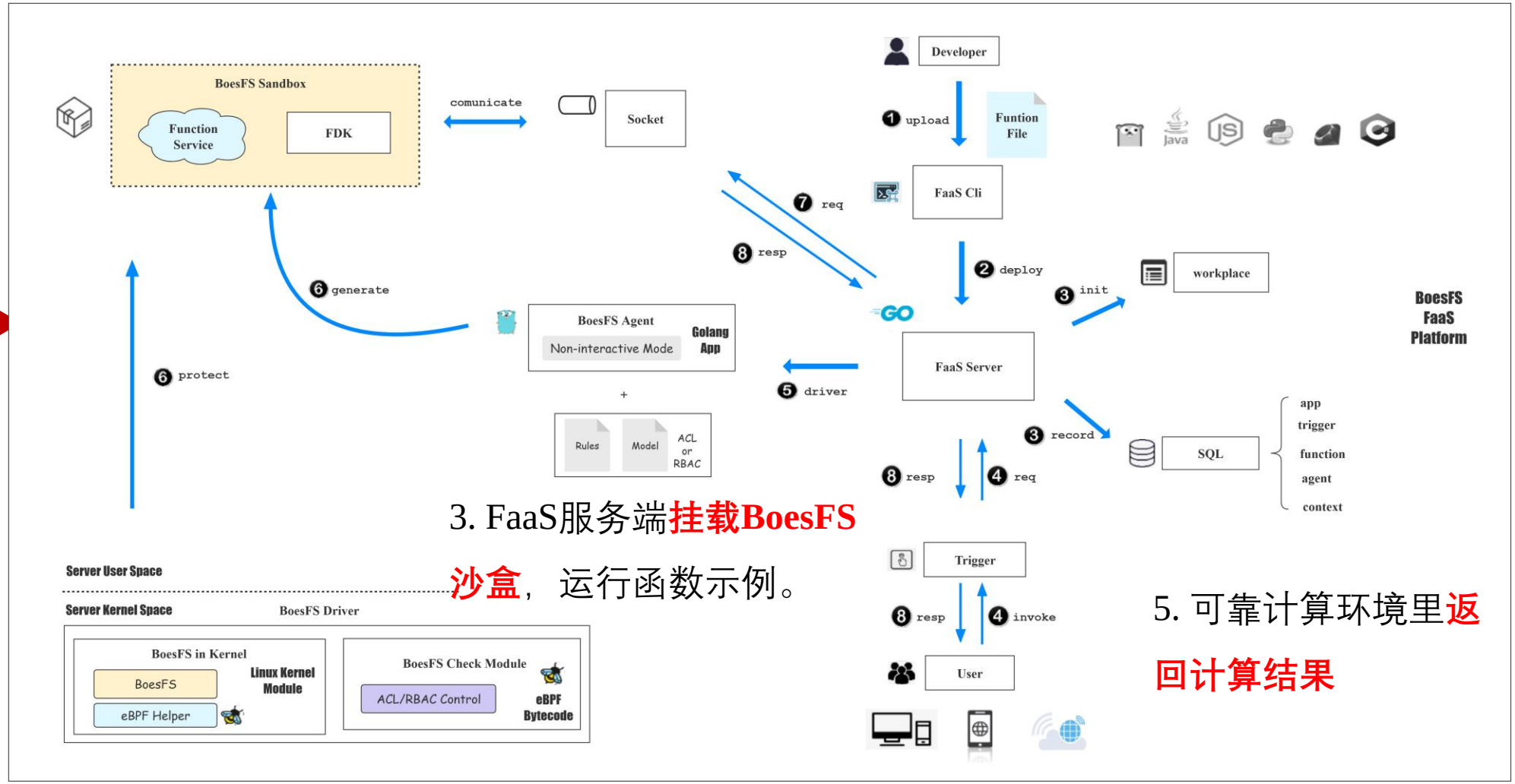


- BoesFS-Agent(Go)**生成沙盒环境**
- 在沙盒里开启一个bash，通过bash运行FDK，生成函数实例
- 函数运行时始终受**BoesFS的强制保护**
- 等待fdkDown信号，关闭沙盒层

3.5 BoesFS FaaS 设计

BoesFS沙盒的接入：

1. 开发者**部署FaaS函数**到FaaS服务端



3. FaaS服务端**挂载BoesFS沙盒**，运行函数示例。

4. BoesFS沙盒根据安全规则**拦截非法请求**。

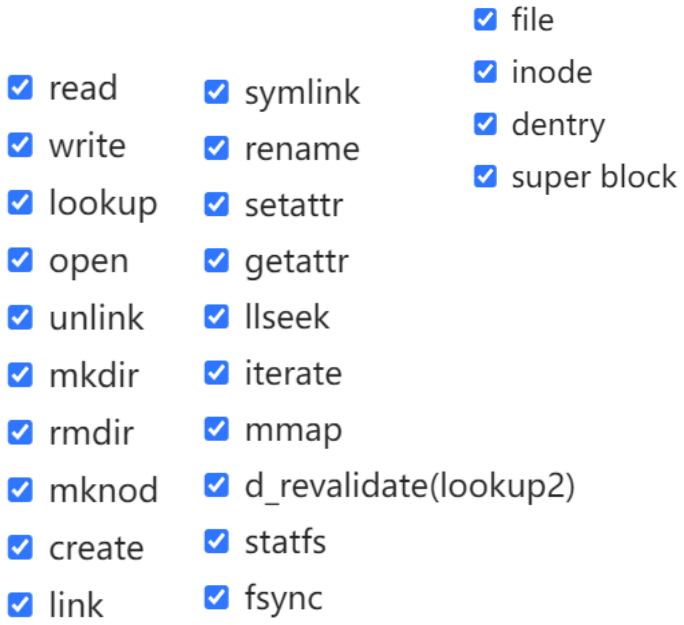
2. 客户适时**触发FaaS函数**

5. 可靠计算环境里**返回计算结果**

4.1 BoesFS功能测试

BoesFS功能测试的主要测试点如下:

- 正确拦截VFS所有文件请求
- ACL模式自定义规则
- RBAC模式自定义规则
- 参数匹配
- 运行时动态替换规则



针对不同的测试点，编写了系列测试用例

BoesFS实现了项目目标预期的全部功能

测试样例	主要测试点	通过情况
基础测试	ACL模式 + VFS操作拦截正确性	PASS
动态替换测试	动态替换规则的正确性	PASS
参数测试	拦截设定的参数正确性	PASS
RBAC基础测试	主体RBAC模式正确性	PASS
RBAC参数测试	客体及参数RBAC模式正确性	PASS

4.2 BoesFS-FaaS响应与性能测试

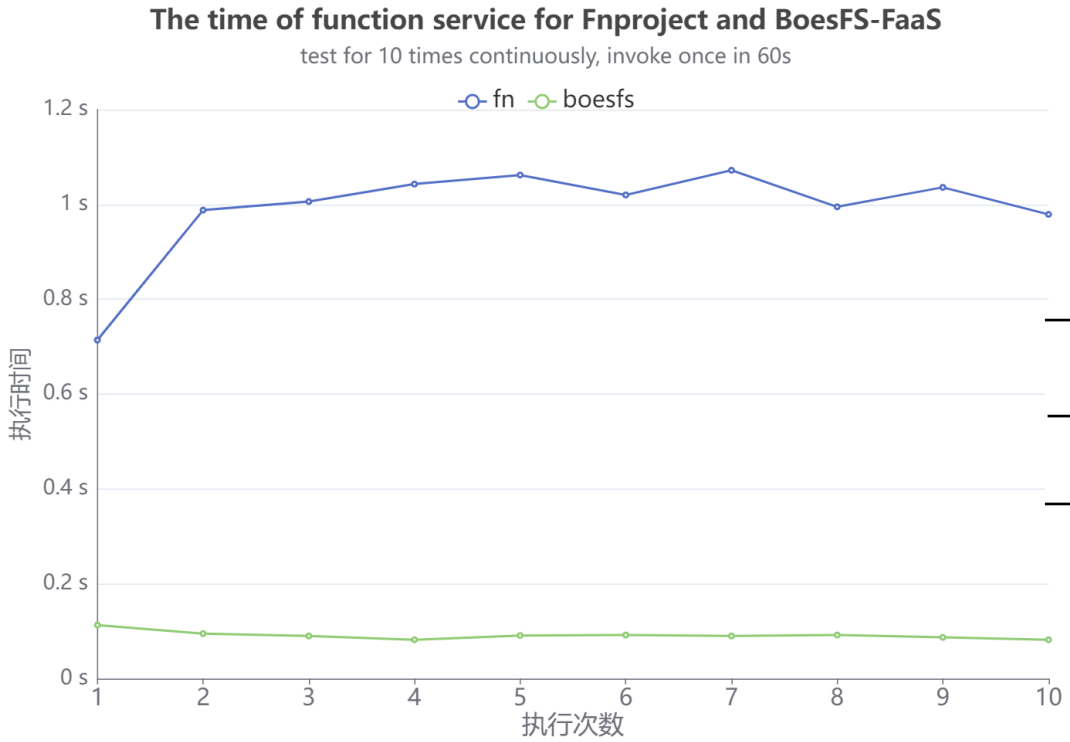
选取了如下指标进行测试：

- 单个函数服务响应的时间
- 高并发条件下的资源消耗

在测试对象上，包括如下两个测试对象：

- Fnproject (基于容器的FaaS)，Fn
- BoesFS-FaaS (基于BoesFS的FaaS)，BoesFS

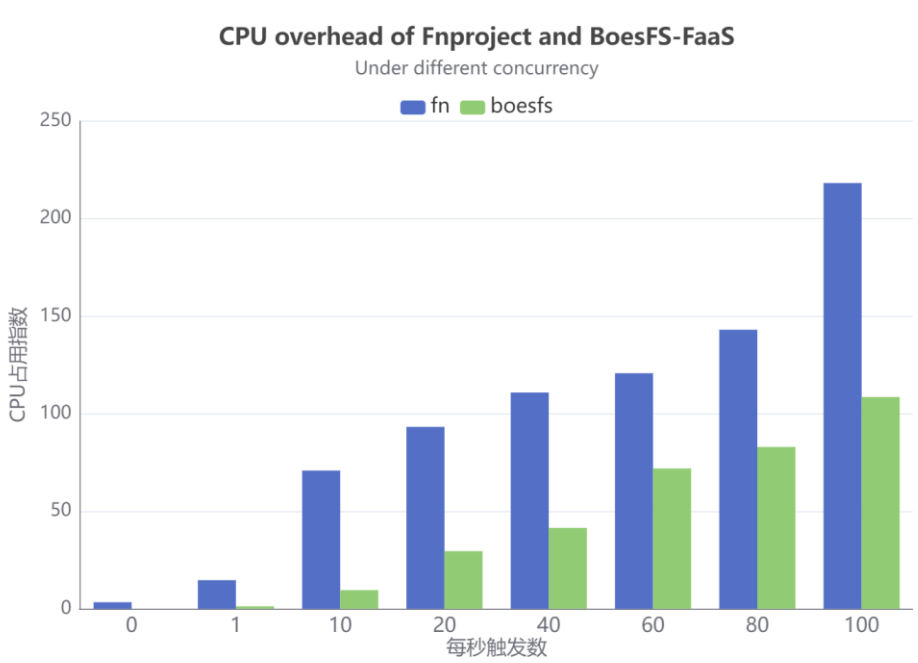
单个函数服务的响应时间：



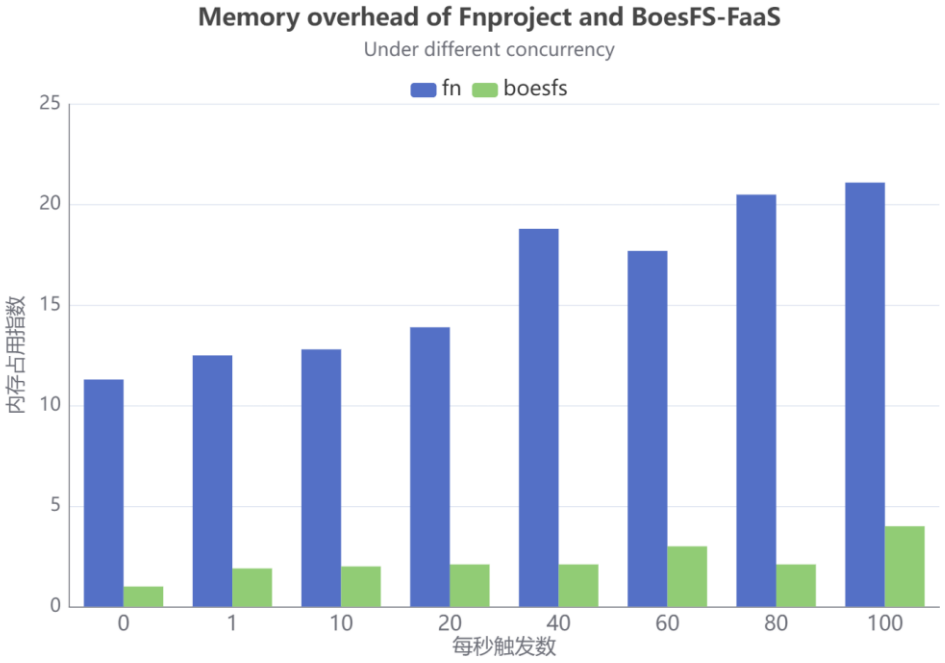
指标	BoesFS	Fn
冷启动平均耗时	0.091	0.991

4.2 BoesFS-FaaS响应与性能测试

高并发条件下的资源消耗：



CPU Overhead



Memory Overhead

4.3 BoesFS文件系统测试

测试方法上，采用**filebench**分别进行了文件系统测试

在性能指标的选择上，我们考虑如下三个性能测试指标:

- 每秒流操作数（ops/s）
- I/O总带宽（Mb/s）
- 流操作响应时间（ms/op）

在测试对象上，包括如下两个测试对象:

- Native(Ext4)环境
- BoesFS环境

4.3 BoesFS文件系统测试

总体平均指标分析：

指标	Native	BoesFS	开销
平均流操作响应时间(ms/op)	5.69	6.00	5.48%
平均带宽(mb/s)	775.55	736.74	5.01%
平均每秒流操作数 (ops/s)	37595.5051	35711.3798	5.01%

从三种指标的对比来看，BoesFS带来的开销均只在5%左右

4.4 BoesFS性能测试

测试方法上，在**三种实际场景下**分别进行了性能测试，比较任务耗时情况

在实际场景的选择上，我们考虑如下三个测试场景：

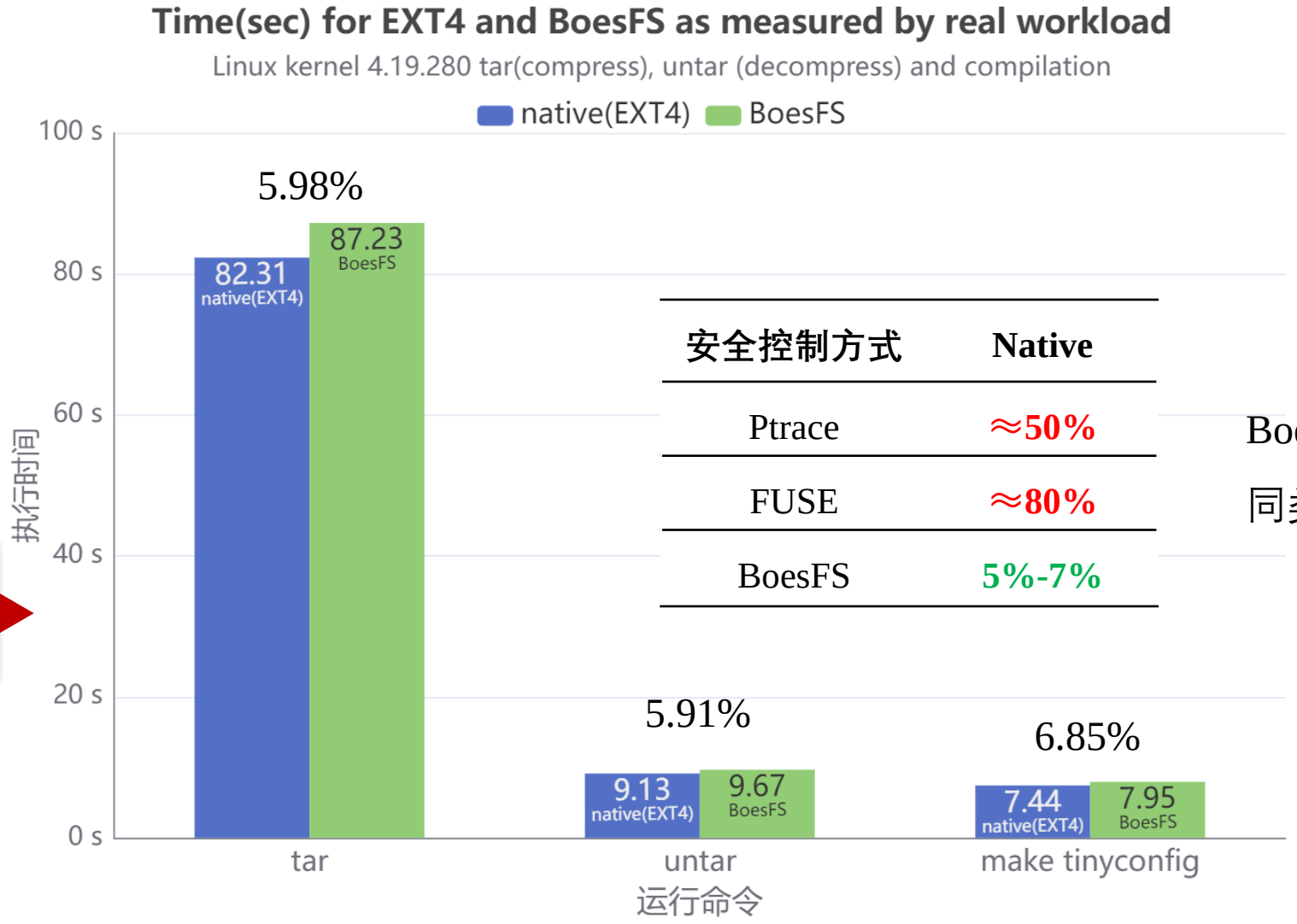
- **压缩内核源码测试**
- **解压缩内核源码测试**
- **编译内核源码（tiny config）测试**

注：内核源码采用Linux 4.18

在测试对象上，包括如下两个测试对象：

- **Native(Ext4)环境**
- **BoesFS环境**

4.4 BoesFS性能测试



安全控制方式	Native
Ptrace	≈50%
FUSE	≈80%
BoesFS	5%-7%

BoesFS沙盒开销低于同类安全访问控制方式

三种具体场景下，BoesFS带来的性能开销均只在7%左右

4.5 BoesFS项目展示


```python
def deldir(dir):
    try:
        if not os.path.exists(dir):
            return "Fail"
        if os.path.isfile(dir):
            os.remove(dir)
            return "Success"
        for i in os.listdir(dir):
            t = os.path.join(dir, i)
            if os.path.isdir(t):
                deldir(t)
            else:
                os.unlink(t)
        if os.path.exists(dir):
            os.removedirs(dir)
        return "Success"
    except:
        return "Fail"
```

恶意函数

```
boes@debian:~/app/del_test$ cat ~/.boesfs/acl/model/model.txt
[request_definition]
r = obj, act

[policy_definition]
p = obj, act

[policy_effect]
e = !some(where (p.eft == deny))

[matchers]
m =  r.obj == p.obj && r.act == p.actboes@debian:~/app/del_test$
```

```
p, /home/boes/test, unlink, dir,  deny
p, /home/boes/test, rmdir,  dir,  deny
p, /home/boes/test, unlink, file, deny
p, /home/boes/test, rmdir,  file, deny
```

ACL/RBAC安全检查规则

拦截成功

```
native@debian:~$ curl -X "POST" -H "Content-Type: application/json" -d '{"p
ath":"/home/boes/test"}' http://192.168.171.155:8080/invoke/01H7VJFWKER2MAQ
6RZJ0000002
{"message": "Fail"}native@debian:~$
```

拦截恶意函数成功

```
boes@debian:~$ tail -f ~/.boesfs/log/python3_19748.txt
[allow] python3,llseek,/home/boes/test/111.txt,0,1
[allow] python3,llseek,/home/boes/test/111.txt,0,1
[allow] python3,getattr,/home/boes/test/111.txt
[allow] python3,read,/home/boes/test/111.txt,8192,0
[allow] python3,open,/home/boes/test/111.txt
```

文件请求的审计

```
boes@debian:~/boesfs/Code$ cd ~/test/
boes@debian:~/test$ cat 222.txt
cat: 222.txt: Operation not permitted
boes@debian:~/test$ cat 222.txt
cat: 222.txt: Operation not permitted
boes@debian:~/test$ cat 222.txt
boes@debian:~/boesfs/Code$
```

kill

终止异常进程

```
boes@debian:~/boesfs/Code$ vim ~/.boesfs/acl/model/policy.txt
boes@debian:~/boesfs/Code$ boesfs -d /home/boes -k 0 /bin/bash
Welcome to BoesFS-Agent!
```

普通用户

非特权用户使用

核心实现：

- 基于**eBPF**和**堆叠式文件系统**实现**轻量级堆叠式沙盒层**，并基于此进一步实现**轻量级且允许自定义安全的FaaS平台**。

- **强安全上**，使用**eBPF字节码**实现**ACL/RBAC访问控制模型**，采用**namespace**保证沙盒的**进程粒度**，设计**动态规则替换**能力，采用**eBPF Map**装填**自定义运行时检查规则**，**沙盒文件系统拦截非法请求**，提供实现强安全防护。

- **高性能上**，基于**小而精的eBPF字节码**和**轻量级的堆叠式文件系统**，并加以**字节码的ACL查询优化设计**，带来了很低的性能开销。

哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

2023年全国大学生计算机系统能力大赛操作系统设计赛作品BoesFS 技术分享

Thanks

BoesFS: 基于eBPF实现的文件系统沙盒及其应用

队伍名称： BoesFS

技术主讲： 杨大荣

辅导老师： 夏文、李诗逸