# ARMlock: Hardware-based Fault Isolation for ARM

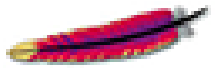
Yajin Zhou, Xiaoguang Wang, Yue Chen, and Zhi Wang

North Carolina State University
Xi'an Jiaotong University
Florida State University

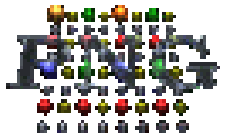
# Software is Complicated and Vulnerable

17 million SLOC

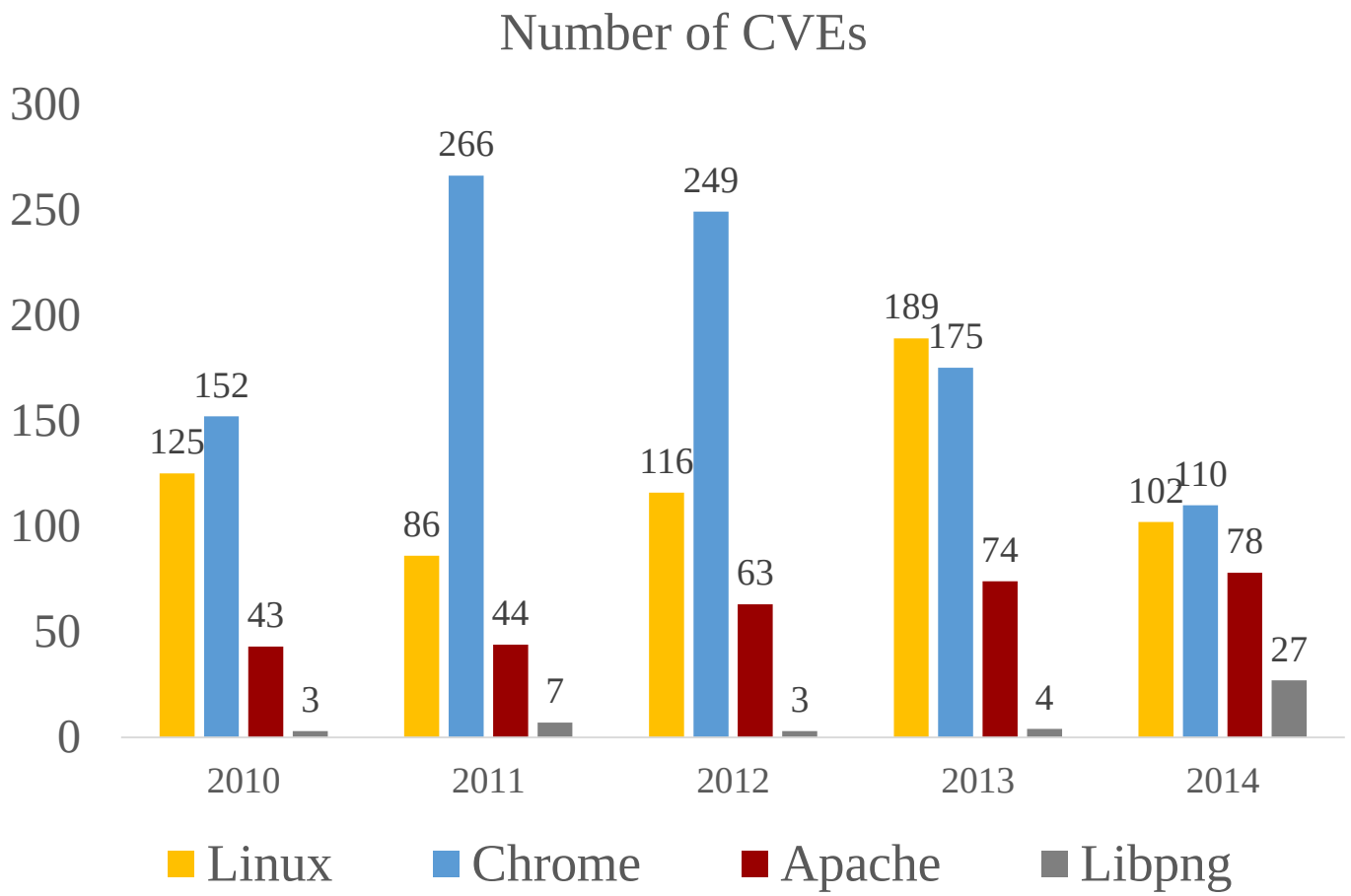
15 million SLOC



2 million SLOC



400 thousands SLOC



# Software is Complicated and Vulnerable

- Code: different sources
  - Third-party libraries, plugins …
- Vulnerabilities in one module could compromise the whole application

Heartbleed

# Software Fault Isolation

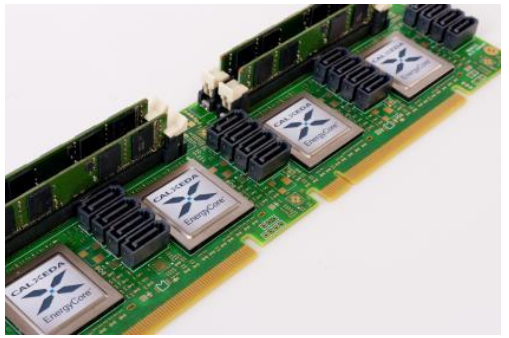
- SFI: security by isolation
  - Split application into different fault domains
  - Separate each domain from others
  - Compromised fault domains cannot affect others
- Widely used in x86 systems
  - Linux kernel: LXFI
  - User level applications: Native client, Vx32 …

**Our work focuses on ARM architecture**

# ARM Architecture is Popular

ARM®

750 million Android devices in 2013
99% are based on ARM architecture




ARM is catching up in the data center server market

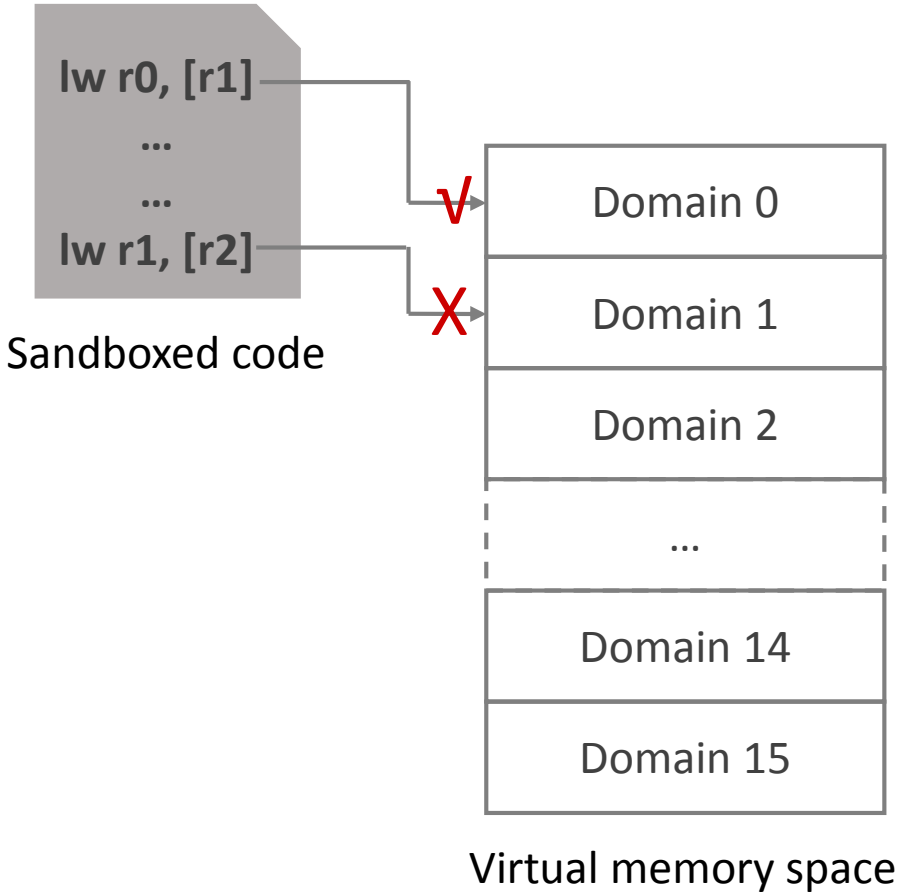
# SFI on ARM Architecture

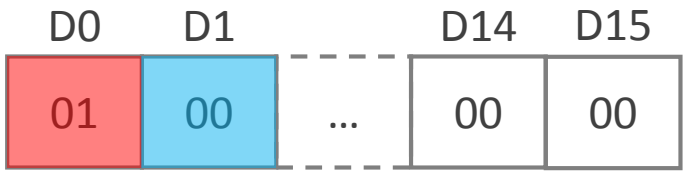
- Native client for ARM
  - Compiler based solution
  - Limitations: assumption on memory layout, hard to efficiently support self-modifying code, and JIT compiling
- ARMor
  - Binary rewriting
  - High performance overhead

# Our Solution: ARMlock

- Strict isolation
  - Memory read/write, code execution, system calls
- Low performance overhead
  - Sandbox context switch, sandbox itself
- Compatibility
  - Memory layout, self-modifying code, JIT compiling
- Leverage an often overlooked hardware feature: Memory domain

# Background: ARM Memory Domain




DACR Register

ARM domain access control

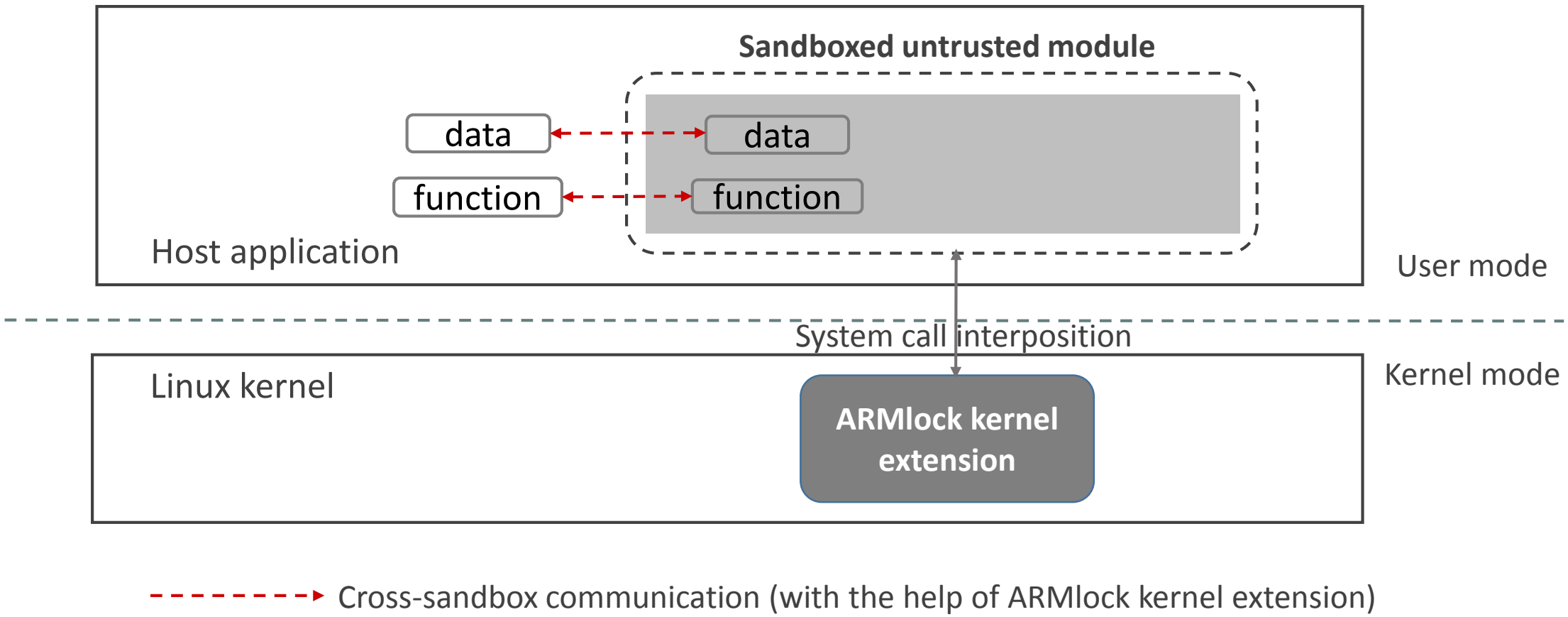
| Type | Value | Description |
|---|---|---|
| No Access | 00 | No access permitted |
| Client | 01 | Permissions defined by page tables |
| Reserved | 10 | Reserved |
| Manager | 11 | No permissions check (unlimited access) |

# Threat Model

- OS kernel is trusted
- Host application is benign but could be vulnerable
- External modules: vulnerable or malicious

**Isolate compromised or malicious modules from the host application**

# ARMlock Architecture



# Sandbox Creation

- Host application asks ARMlock kernel module to create a sandbox
- Kernel module initializes the sandbox
  - Locate first level page table entries
  - Assign different memory domains to the host application and sandboxes
- Memory domain assignment cannot be changed by the sandbox

# Sandbox Switch

- DACR register is saved in the thread control block
- DACR register is updated when switching sandboxes
  - Only current domain (and kernel) are accessible, not other domains
- Multithreading is naturally supported
  - Each CPU core has its own DACR register

# Cross-sandbox Communication

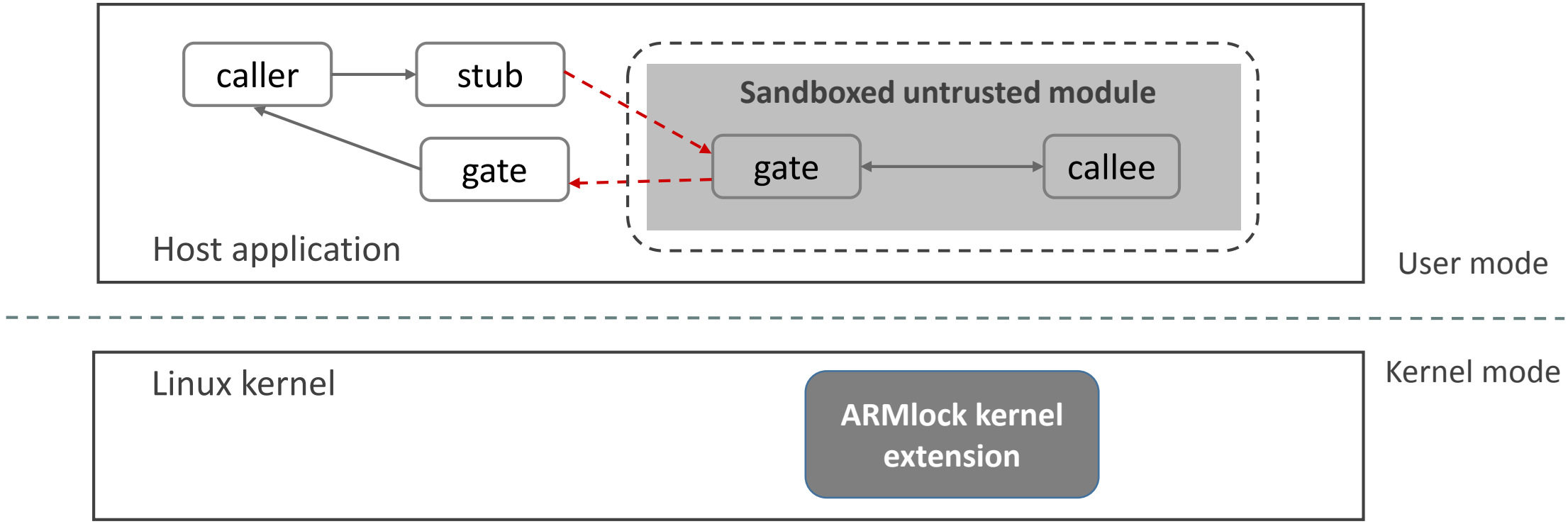
- Inter-module function call
- Inter-module memory reference

# Inter-module Function Invocation

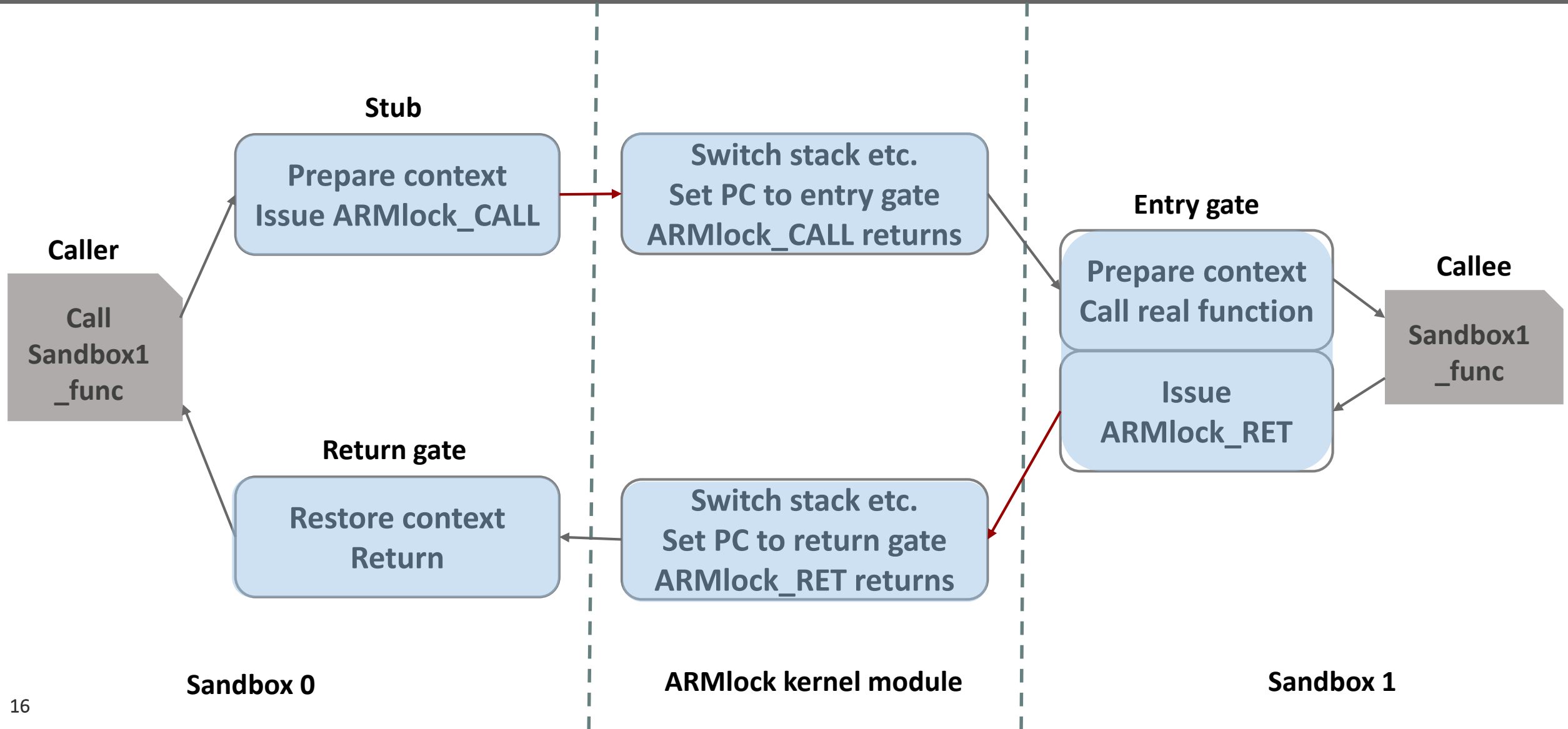
- Two new system calls
  - ARMlock_CALL: inter-module function call
  - ARMlock_RET: inter-module function return

# Inter-module Function Invocation

- Inter-module function invocation



# Inter-module Function Invocation

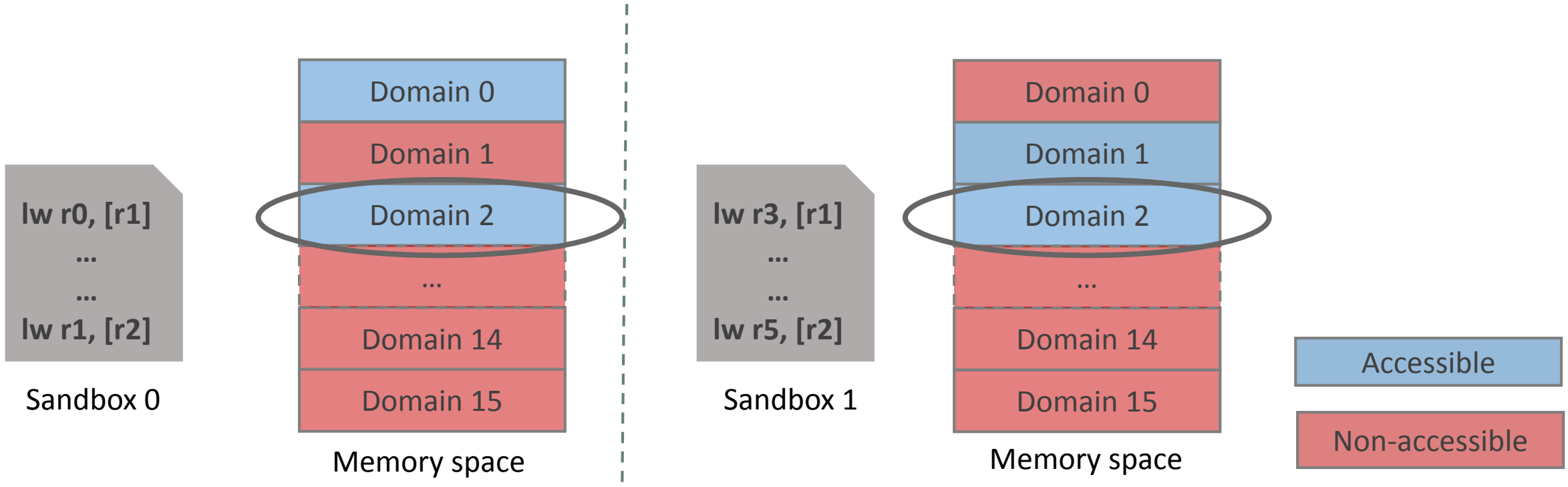


# Inter-module Memory Reference

- Kernel assisted memory copy
  - Kernel marks both domains as accessible
  - Copy data into the destination sandbox
  - Restore the DACR register

# Inter-module Memory Reference

- Shared memory domain: using a domain which is accessible in both sandboxes
- Data from sandboxed modules should be sanitized



# System Call Interposition

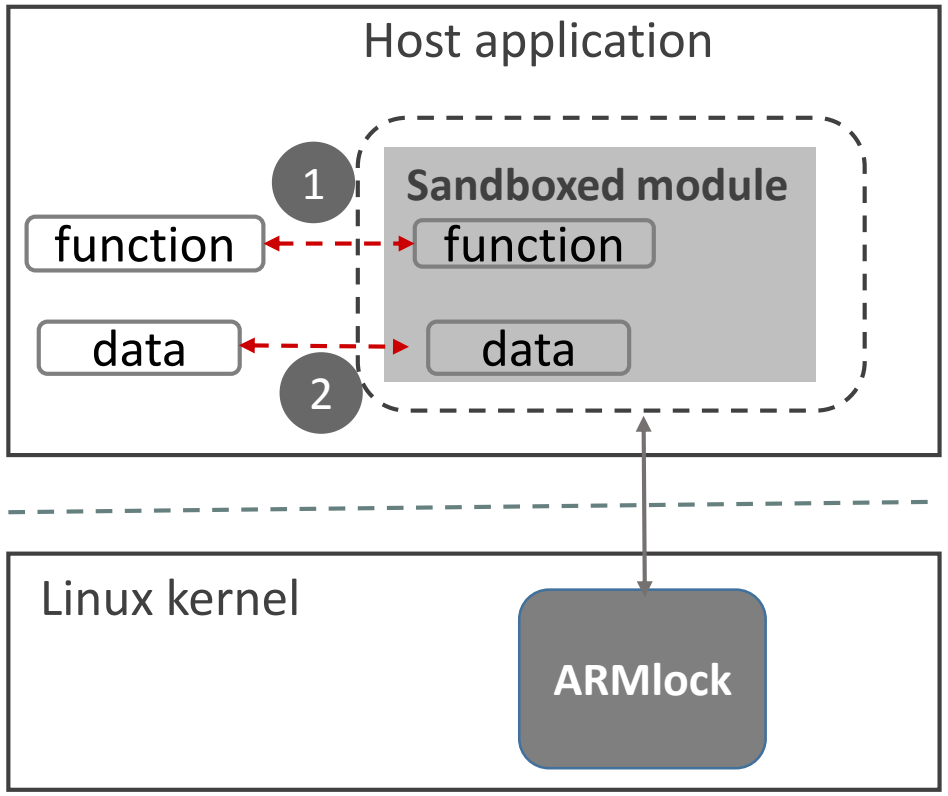
- Recent Linux system has 380+ system calls
  - Normal applications may use less than that, e.g., around 60
  - More system calls may expose more kernel vulnerabilities
- Host applications in ARMlock could control system calls available to sandboxed modules
- Implemented through the seccomp-BPF framework

# Evaluation

- Security analysis
- Performance overhead
  - Sandbox switch latency
  - Sandbox itself

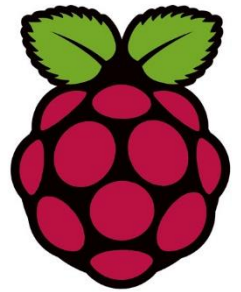
# Security Analysis

- Cross-sandbox communication
  - Inter-module function invocation
  - Inter-module memory reference
    - Kernel assisted memory copy
    - Shared memory domain: race condition



# Performance Evaluation: Configuration

| Item | Configuration |
|---|---|
| CPU | ARM1176JZF-S 700MHz |
| RAM | 512MB |
| OS | Raspbian (based on Debian) |
| Kernel | Linux 3.6.11 |
| LMbench | Version 2 |
| nbench | Version 2.2.3 |

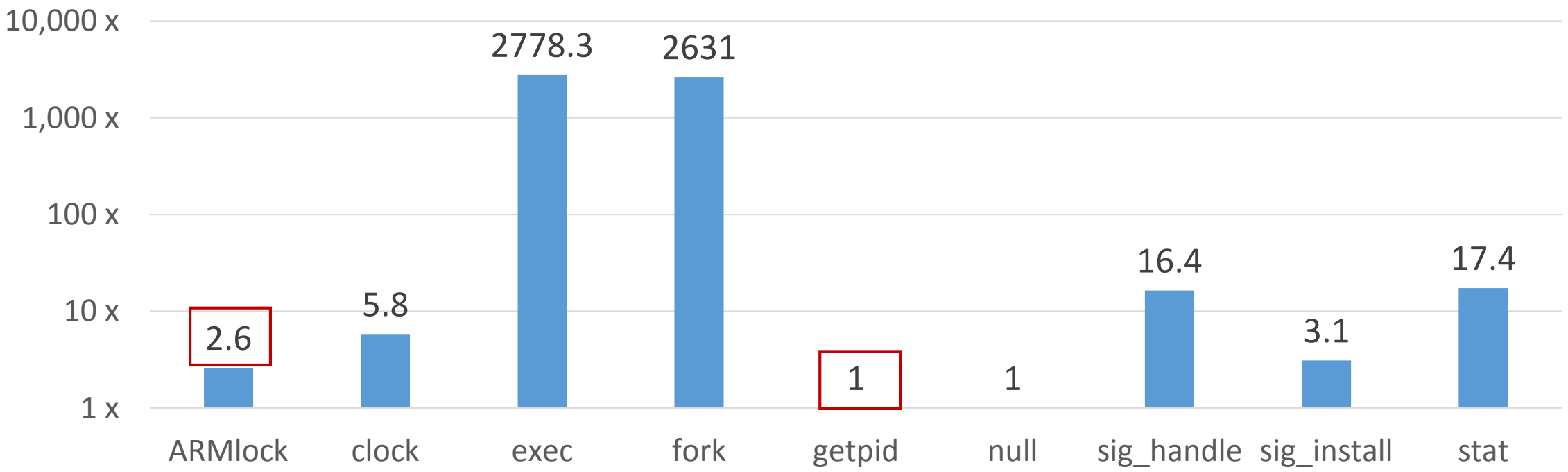


# Sandbox Switch Latency

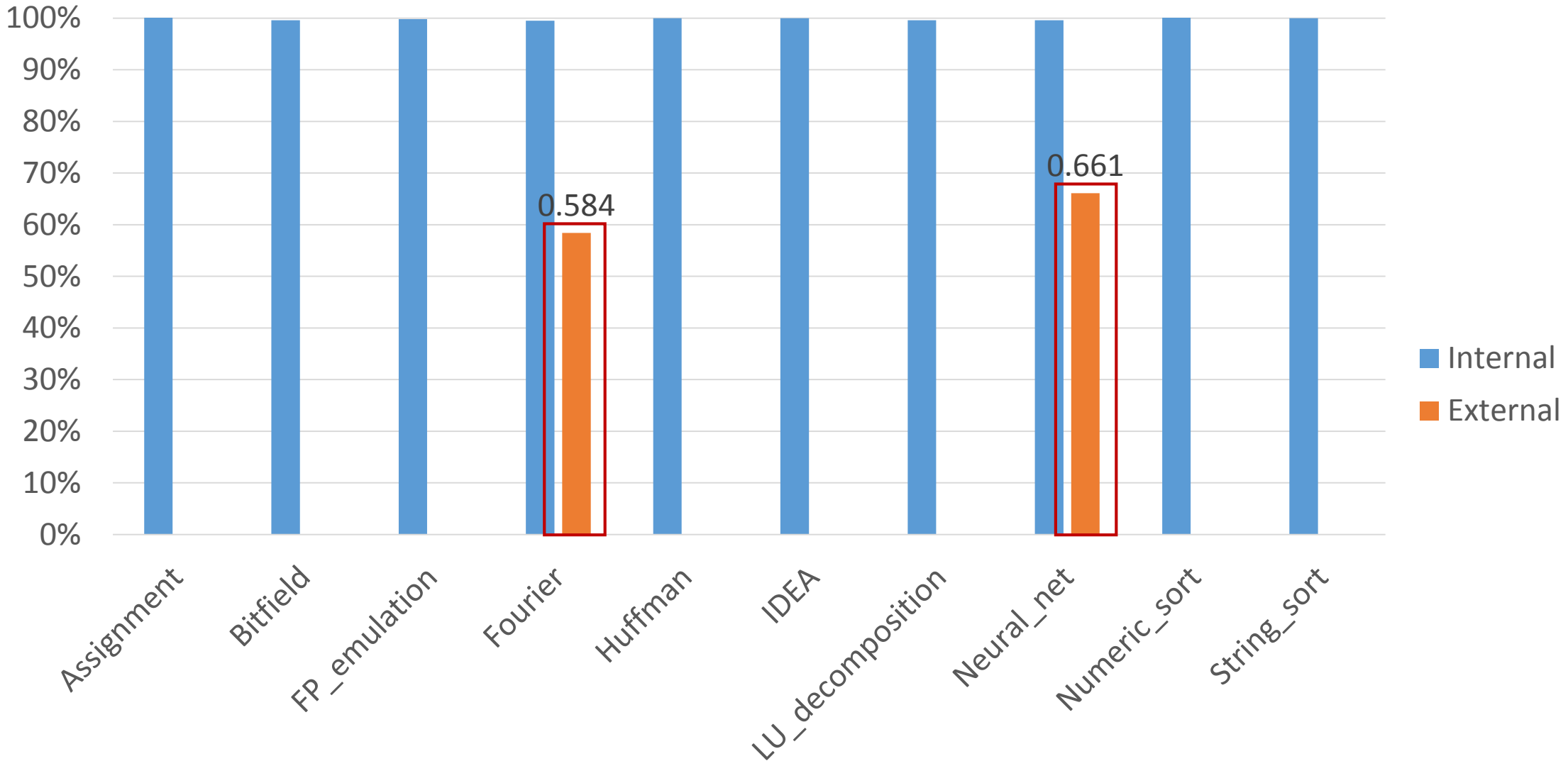
- Call a simple *inc* function inside the sandbox
  - 1 second: 903,343 inter-module calls -- 1.1 μs for each call

# Sandbox Switch Latency

- One sandbox switch: two system calls



# Performance Overhead



# Discussion

- Some developer efforts are required
  - Refactor the application into domains
  - Avoid frequent domain switch
- Need to use short format page table in latest ARM architecture
- Kernel-level sandbox
- Other OS support

# Takeaway

- ARMlock: a hardware-based fault isolation for ARM
  - Strict isolation
  - Low performance overhead
  - Better compatibility

# Q&A