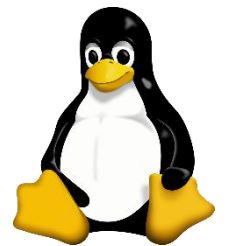
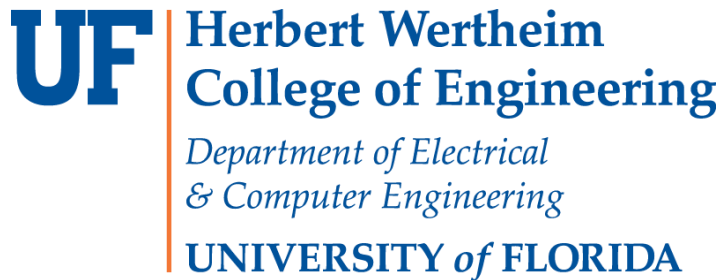


A Modular Approach To Power Management Fuzzing

Linux Plumbers Conference 2025, Tokyo Japan

Darrion Ramos, Dr. Tuba Yavuz, Dr. Bai Yihang, Victoria Siver



Overview

- Motivation and Background
- Challenge #1: Low Power Coverage
- Challenge #2: Input Device
- Proposed Architecture (S2E)
- S2E Demo
- Proposed Architecture (Physical)
- Physical Demo
- Summary
- Future Work

Motivation and Background

Problems

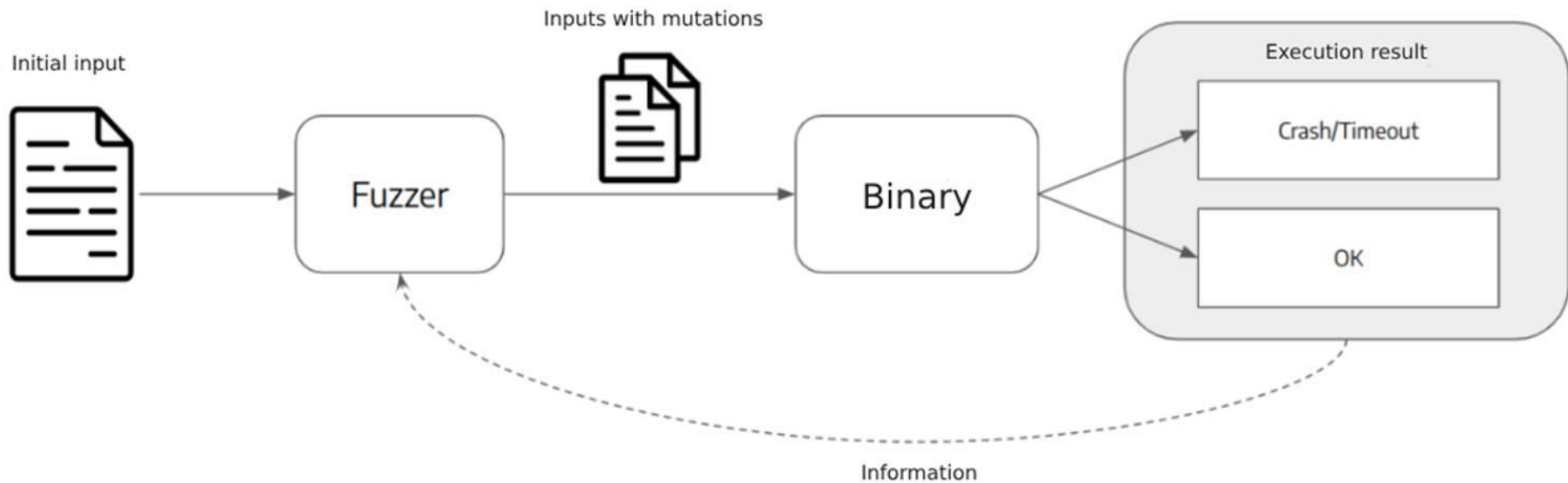
- Low power state observability
- Fault reproduction
- Kernel Power Management (PM) profiling difficulty
 - Failures at suspend/resume
 - Timing and race conditions
 - System interdependencies (USB, HID, etc)
- Tracing and failure analysis, but no fault space exploration

Goals

- Create a fuzzer to explore the PM bug space
 - Fault reproducibility
 - **Low power state tracing for failure analysis**
 - Modular (mutators and targets)
- Start with simple USB and PM interactions
 - Identify races and gain insight
 - Remote wakeup

What is a Fuzzer?

- Creates inputs to explore target program code space
- Generates inputs based on previous run coverage
- Saves inputs on crash to reproduce



Real PM-USB Bugs

- January 2024: a [patch](#) for DWC3 suspend being blocked from incorrect dwc->connected flag behavior
- August 2024: a [patch](#) for improper polling behavior of the DSTS link state which resulted in inconsistent states between the host and the gadget

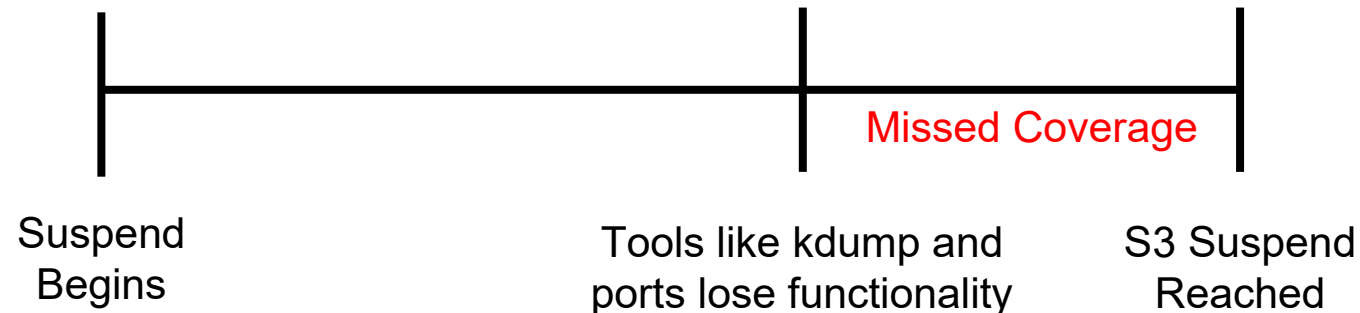
Existing Kernel Fuzzing Solutions

- Syzkaller
- kAFL
- Unicore
- USB fuzzing possible, but PM integration difficult
 - Design choices do not account for events like suspend
 - Kernel setups are limited and fragile
 - Later discovered that VMs are not capable of proper PM emulation required

Challenge #1: Low Power Coverage

Existing Tools

- CONFIG_PM_DEBUG
 - Identify culprit drivers, no analysis
- TRACE_RESUME with S2RAM
 - Uses the Real Time Clock (RTC) to save culprit's magic number
- Pstore/kdump
 - Does not generate information from mid-suspend hangs
- Serial port with no_console_suspend
 - Decent coverage up to suspend, but still loses power



Syzkaller Extension

- Initially tried to modify syzkaller's USB fuzzer
 - Additional overhead
 - Focused on enumeration
 - PM integration not trivial
 - Maintaining system functionality after suspend
 - Moving target to another device

Custom QEMU/KVM Fuzzer

- Traditional methods have poor low power state coverage
- RTC can keep needed coverage, but not in a VM
- Hook into QEMU basic blocks
 - Outside of the kernel means full coverage during low-power states!

S2E Extension

- Basic block coverage of the KVM
- Easy to modify with extensions
- Tradeoff of added overhead for faster prototyping

Challenge #2: Remote Wakeup

Input Devices (USB Emulation)

- Raw-gadget
 - Allows for remote wakeup from an external device using USB On The Go (OTG)
 - Simple
- Facedancer
 - More flexibility and control
 - Higher Complexity

Raw-gadget Modifications

- Added support to issue remote wakeup on detection of host suspension
- Added support for more keyboard events

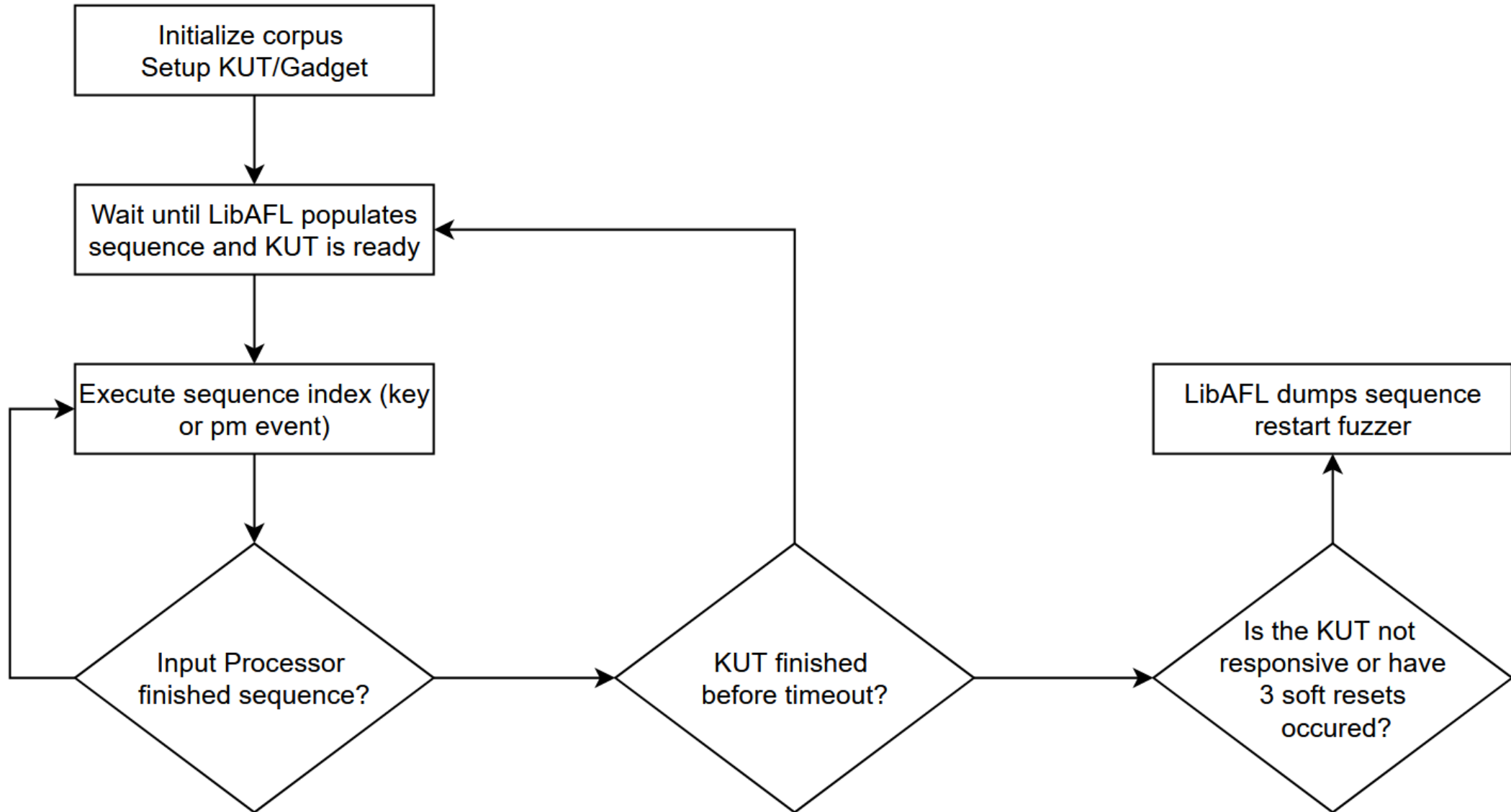
```
pm_usb / raw-gadget / raw_gadget / raw_gadget.c
1292 }
1293
1294 static int raw_ioctl_remote_wake(struct raw_dev *dev, unsigned long value)
1295 {
1296     struct usb_gadget *g = dev->gadget;
1297     int rv;
1298
1299     if (value != 0)
1300         return -EINVAL;
1301
1302     rv = usb_gadget_wakeup(g);
1303     INFO(dev, "%s --> %d\n", __func__, rv);
1304     return rv;
1305 }
```

Raw-gadget Findings

- USB 2.0 OTG lacks proper remote wakeup support
 - RaspberryPi Zero failed
 - USB 3.0 devices needed
 - Radxa Rock 4se, BeagleBone Black support it
- DWC3 gadget code bugs result in improper remote wakeup
 - Kernel version 6.15 fixes USB link state (DSTS) polling issue that resulted in inconsistent states between the host and gadget
 - [Fixed in this patch](#)
 - Reproduced on Radxa Rock 4se kernel 5.10

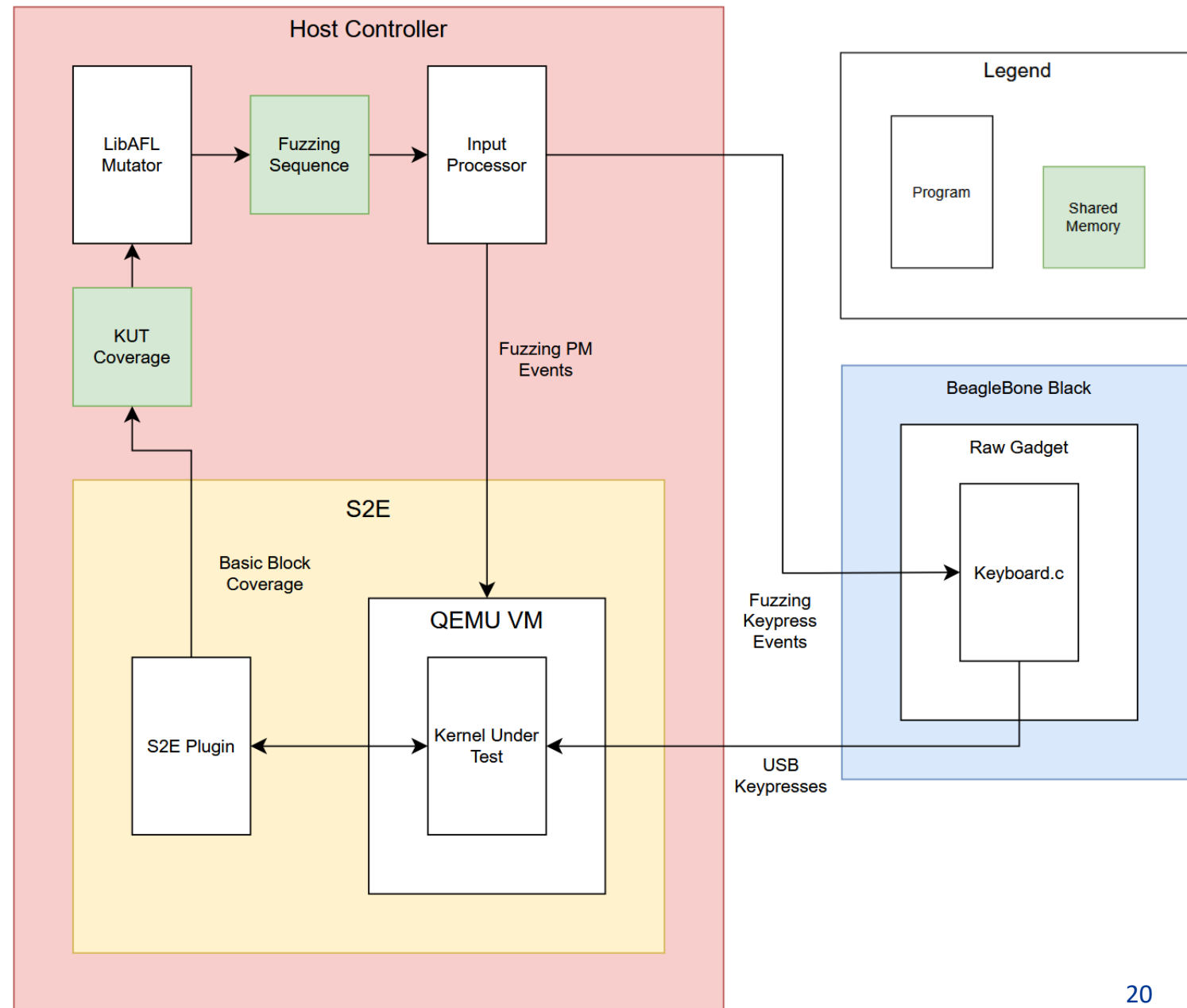
Proposed Architecture (S2E)

Finite-State Machine



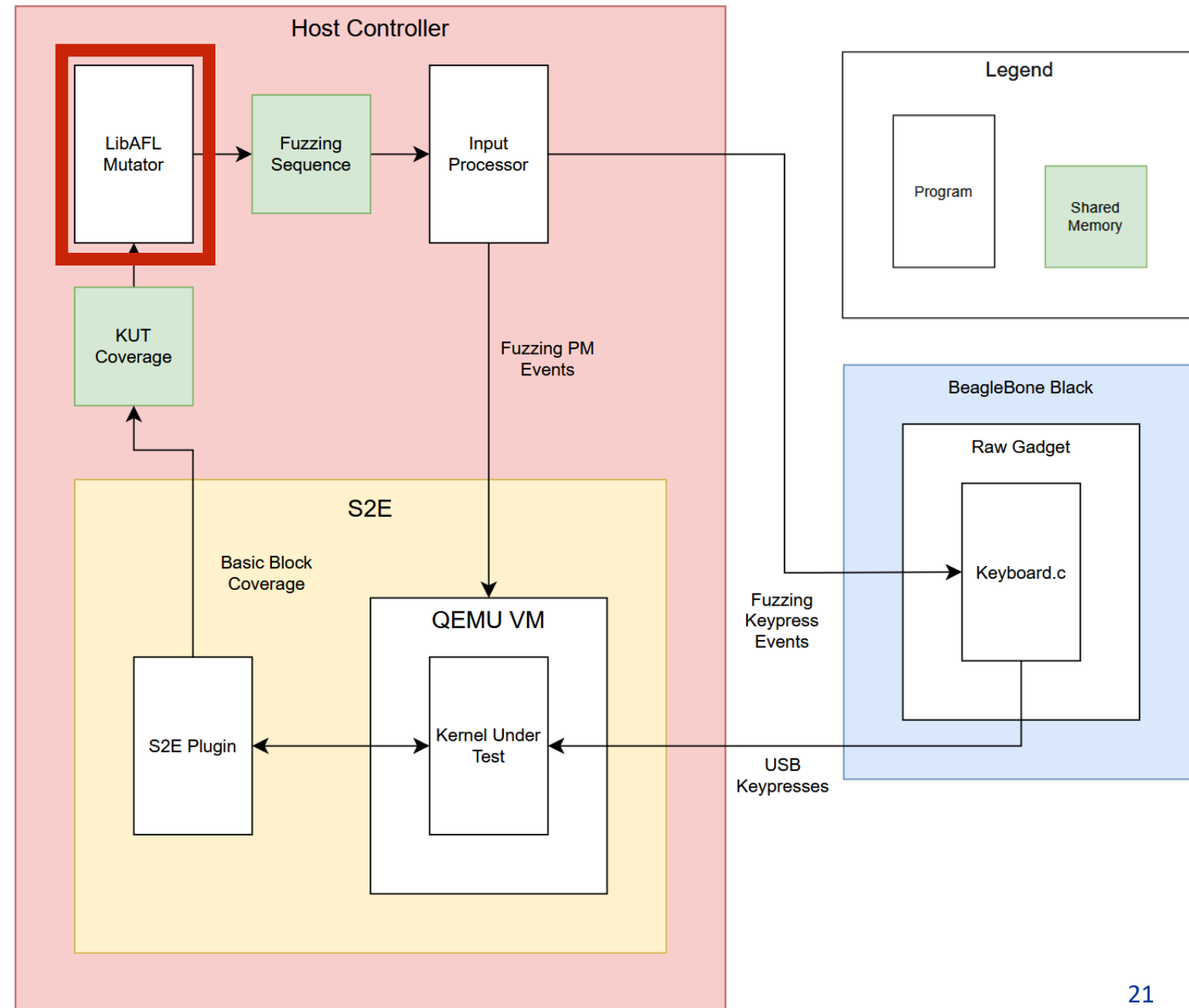
S2E Architecture - High Level

- S2E observes kernel even in low power
- Mutator, coverage, and target independent and modular
- VM provides safe and consistent start
- **VM limits PM functionality greatly**



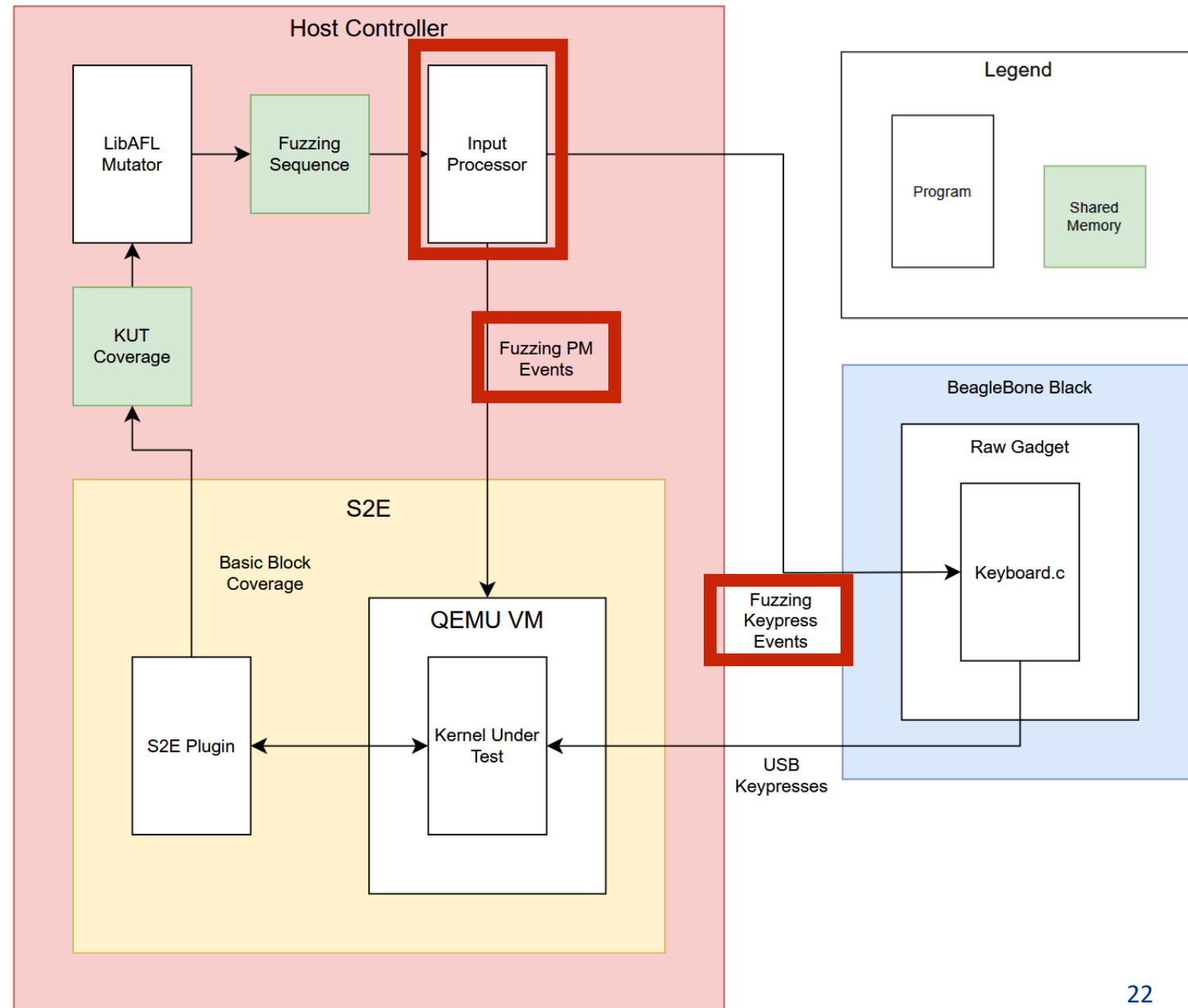
S2E Architecture - LibAFL

- Generates corpus
- Creates byte sequences representing USB and PM events
- Mutations based on Kernel Under Test (KUT) coverage
 - Selective BB coverage



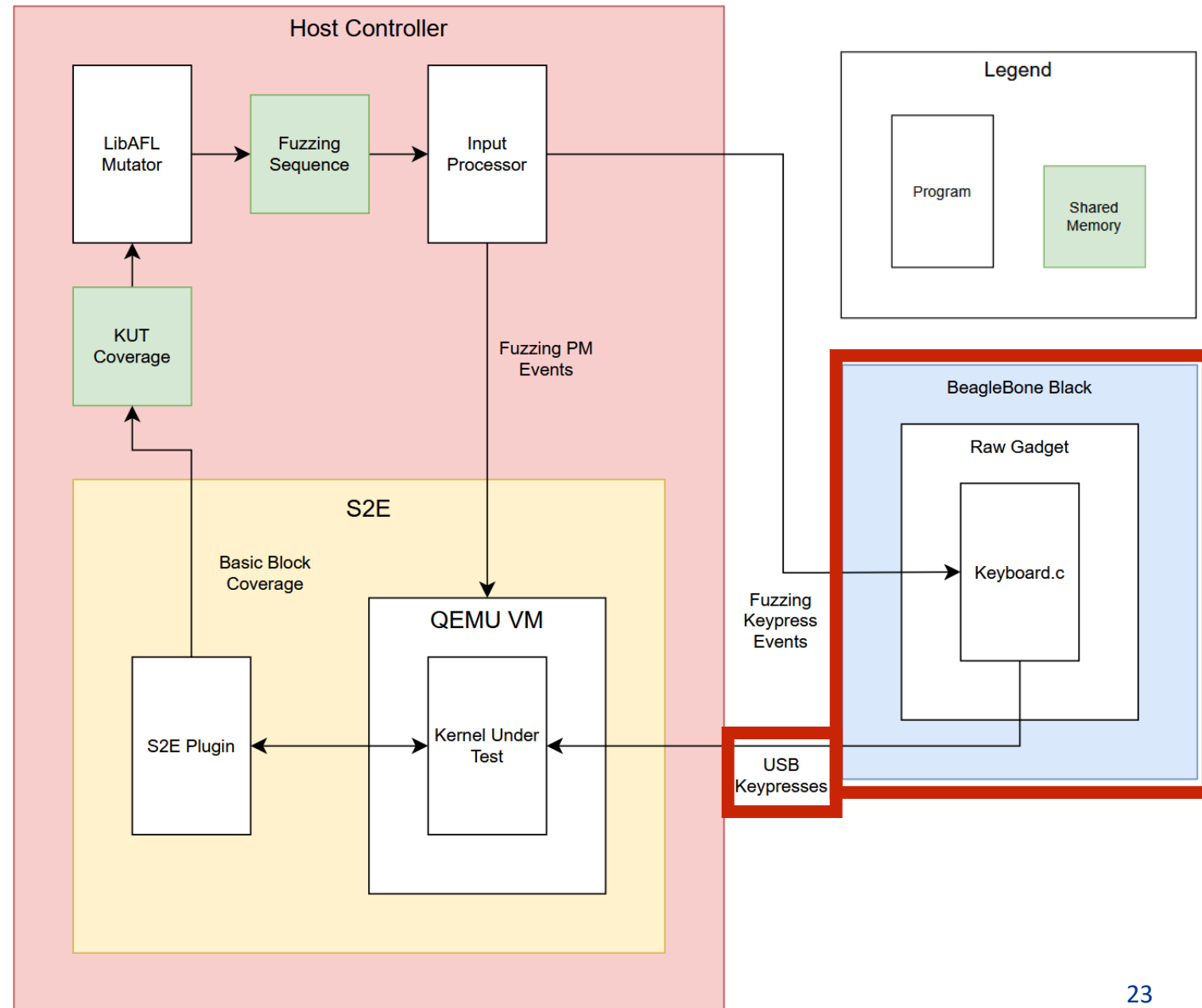
S2E Architecture – Input Processor

- Converts sequence array into actions
- PM events like suspend are issued to the VM
- USB events are sent to an external gadget device



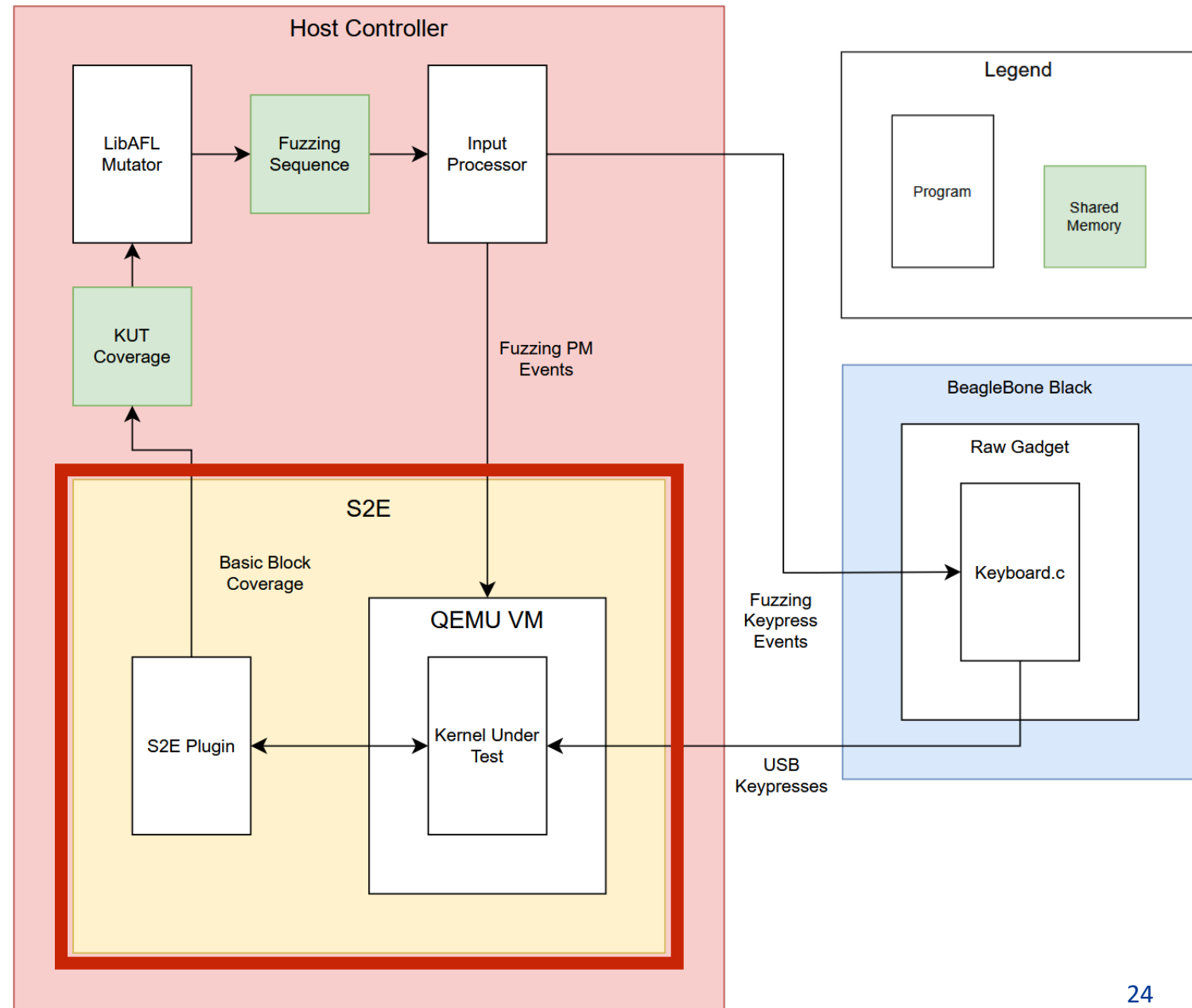
S2E Architecture – Gadget Device

- Leverages a modified raw-gadget
- External device for remote wakeup
 - Automatically wakes host when low power state detected in link
 - **Not supported in VM**
- Processes ASCII from input processor into USB scancodes



S2E Architecture – S2E

- Hosts QEMU/KVM and KUT
- Passes through USB device(s) to VM
- Custom S2E plugin fills coverage map of interesting BBs hit



S2E PM Limitations

- Missing ACPI tables
- USB hubs not able to support remote wakeup
 - Power Management Event capabilities bits are masked
 - USB endpoints suspended
- Relies on external signal from QEMU for wake
 - Kernel does not know what caused the wake

S2E Demo

S2E Pros and Cons

- Pros
 - Best low-power coverage method with BB monitoring
 - Easy extensions of s2e coverage
- Cons
 - **QEMU PM support not sufficient**
 - Significant overhead from BB coverage and plugins reduce kernel speed
 - Kernel image difficult to modify

New Challenges

External KUT Coverage

- Ftrace
 - Hash caller callee pairs of relevant functions into a hash map
 - Send block updates of map to host for coverage guided fuzzing
- Intel Processor Trace
 - Dump CPU instruction data and call trace
 - Do not stop the trace and allow corruption of metadata
 - This allows for the most coverage before a low-power crash
 - Critical file size easily restorable
- RTC
 - Modify kernel to encode critical function coverage into RTC

Clean Reset

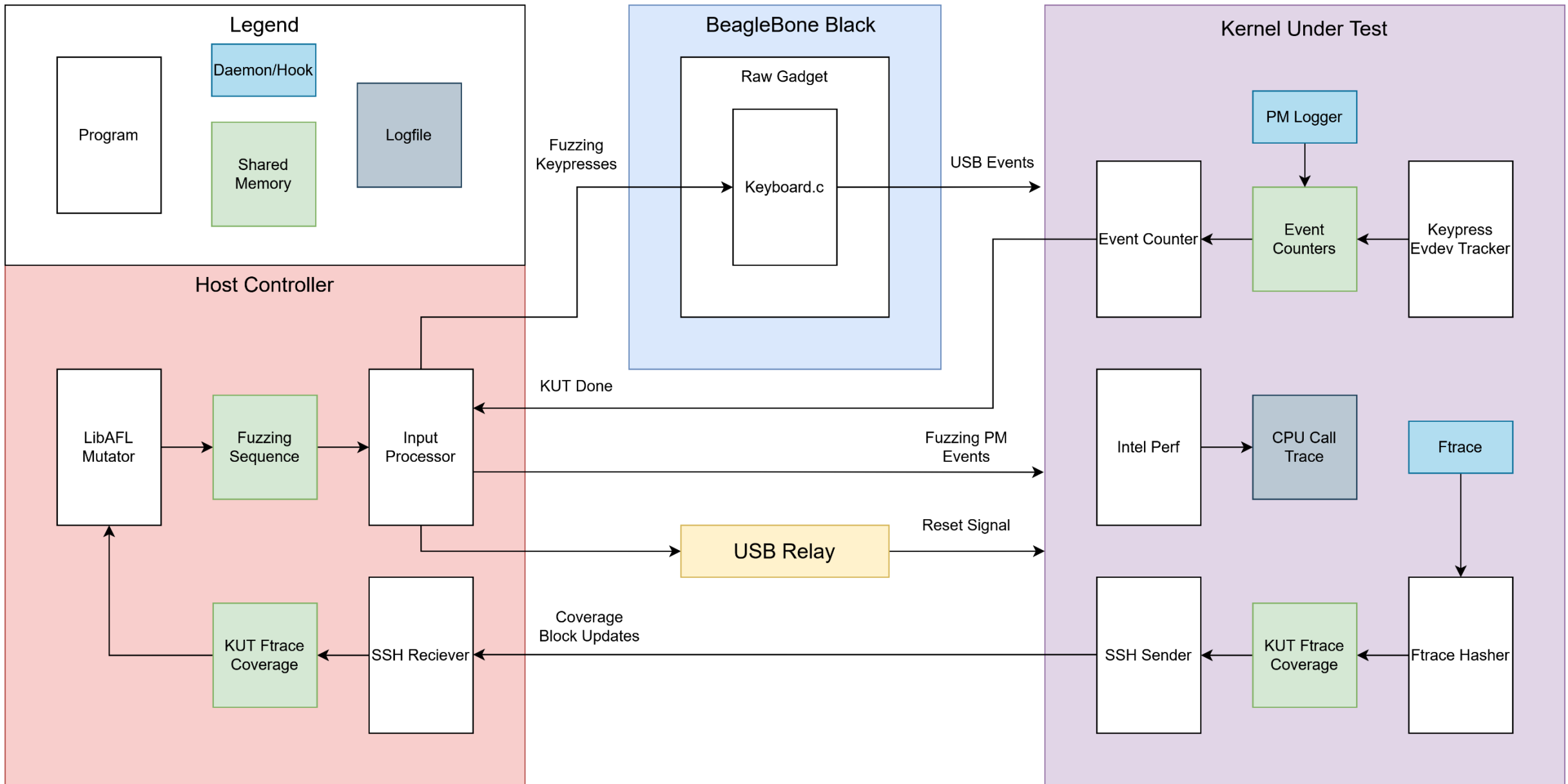
- USB relay connected to the physical reset pins of the device.
 - Power cycle on kernel panic and restart fuzzing loop
- Intel Active Management Technology (AMT) remote management tool
 - Kernel level access to determine crash and reset

Tracking Event Completion

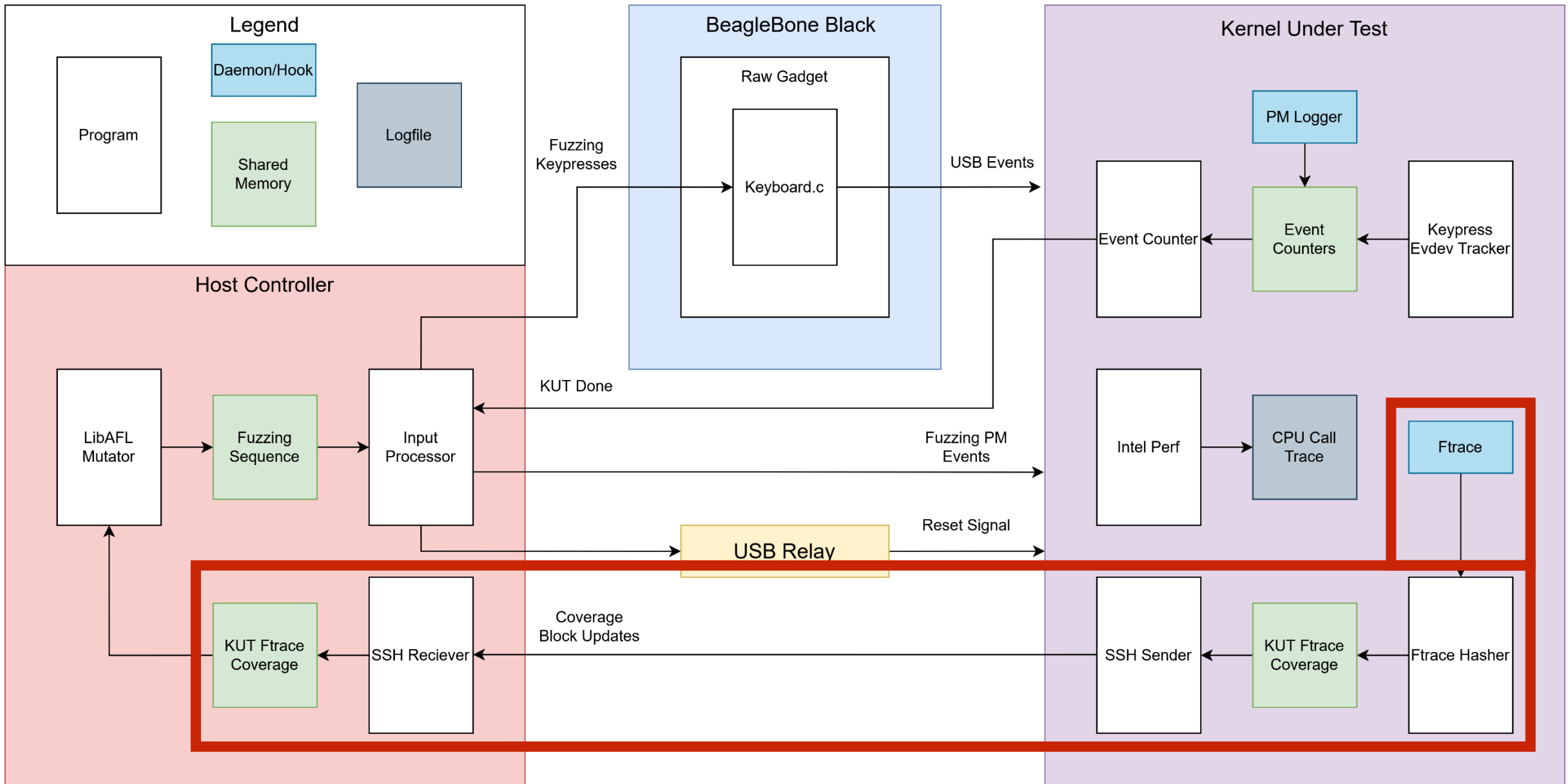
- Feedback to inform host fuzzer the sequence has finished
 - Evdev script to monitor keyboard events
 - Script in pm-utils /etc/pm/sleep.d for power events

Proposed Architecture (Physical)

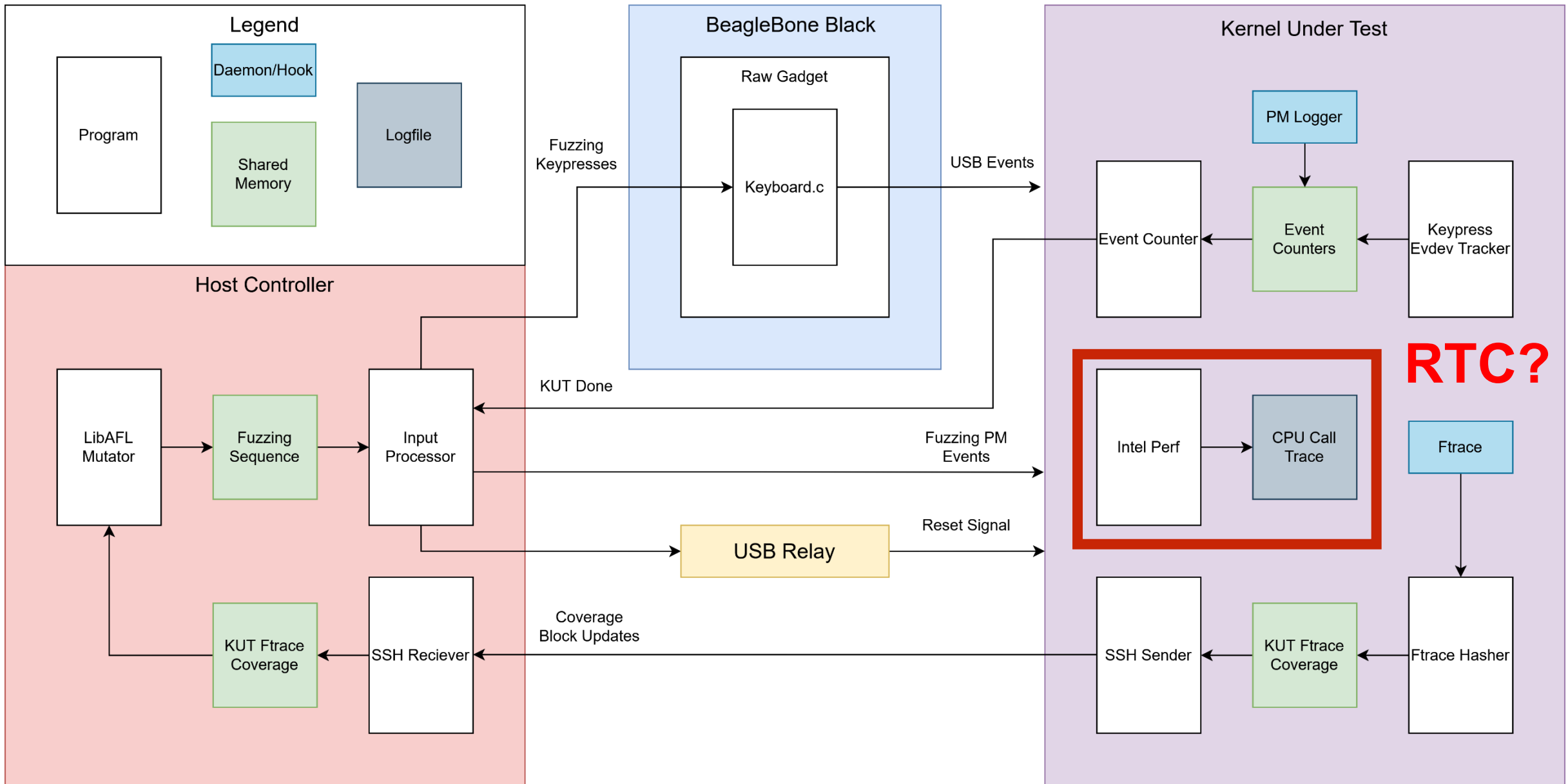
Physical Architecture – High Level



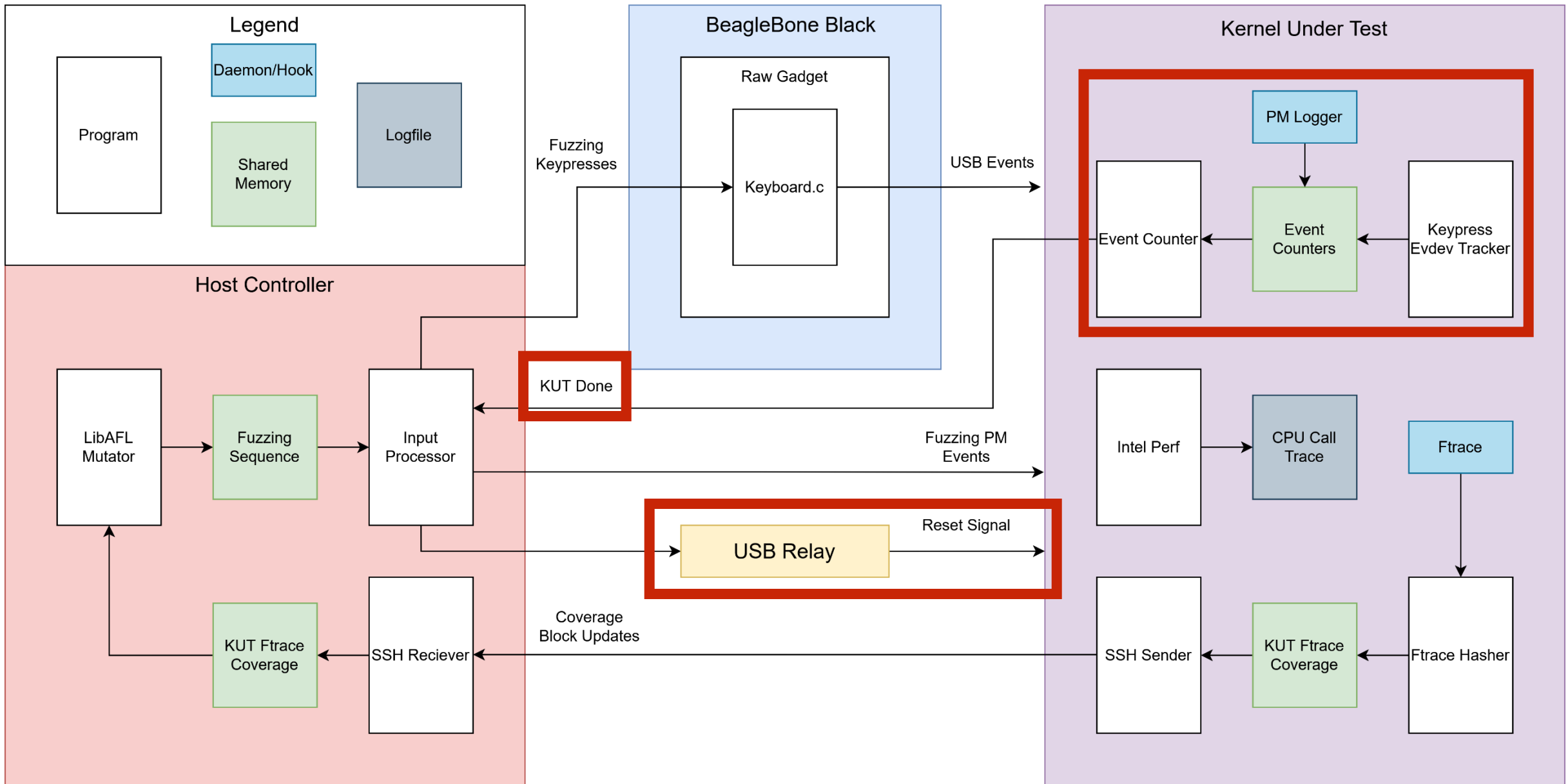
Physical Architecture – Fuzzer Coverage



Physical Architecture – Failure Analysis



Physical Architecture – Control Flow



Physical Architecture – Pros and Cons

- Pros
 - Full PM support, remote wakeup
 - No emulation – real world
 - Faster than VM
- Cons
 - Higher complexity and synchronization cost
 - Not as safe as VM for fuzzing (kernel corruption)
 - Low-power coverage is more difficult

Physical Architecture Demo

Physical Architecture Demo



Results

- Only short testing campaigns ~1hr
- Lock screen bug after fuzzing
 - Unable to enter password without entered keys disappearing
 - Need to use ctrl+alt+f1 a few times to fix
- Reproduced DSTS link bug
- Need to run longer campaigns and analyze traces more

Conclusion - General

- PM code and interactions benefit from more testing
- Existing fuzzers cannot currently target PM events
- QEMU/S2E do not model many needed PM functionalities
 - ACPI, PME generation, remote wakeup

Conclusion - Fuzzer

- This fuzzing setup contributes:
 - Real device/kernel interaction fuzzing
 - Synchronization exploration of power events and devices
 - Low-power state coverage for suspend/resume crash analysis
- Features
 - Mutator, target, and coverage, modularity
 - Timing control
 - Failure recovery

Conclusion - Architectures

- Two different architectures
 - S2E gives basic block low-power coverage but no PM support
 - Physical gives full PM support but coverage is harder

Arch	Low-power code coverage acquisition	Speed	Kernel state safety	PM Support
S2E	Easy	Slow	VM snapshot	Lacking
Physical	Difficult	Fast	Moderate risk	Full

Future Work

- Longer fuzzing and trace analysis
- AMT Remote Management
- Coverage
 - RTC low-power coverage
 - eBPF for added custom coverage points
- New Devices
 - Multiple USB Devices
 - Bluetooth Devices
 - Mobile phone
 - GPU

Thank You!

- Dr. Tuba Yavuz
- Dr. Bai Yihang
- Victoria Siver
- Alan Stern (USB)
- Andrey Konovalov (Syzkaller)
- Vitaly Chipounov (S2E)
- Dominik Maier (LibAFL)

Questions?

Darrionramos@gmail.com

Backup Slides

Existing USB Fuzzing Solutions

- USBFuzz – Driver I/O paths and logic
- FuzzUSB – Gadget state machines and protocols
- While it is possible to extend these, they are not designed for PM/system level interactions

Basic PM-USB Race

- A kernel starts suspending a device while receiving traffic
- This is already accounted for but...
 - Good profiling/fuzzing starting point
 - Complex systems and timings mean more faults possible

