# ReUSB: Replay-Guided USB Driver Fuzzing

**Jisoo Jang**, Minsuk Kang, Dokyung Song
Department of Computer Science
College of Computing
Yonsei University

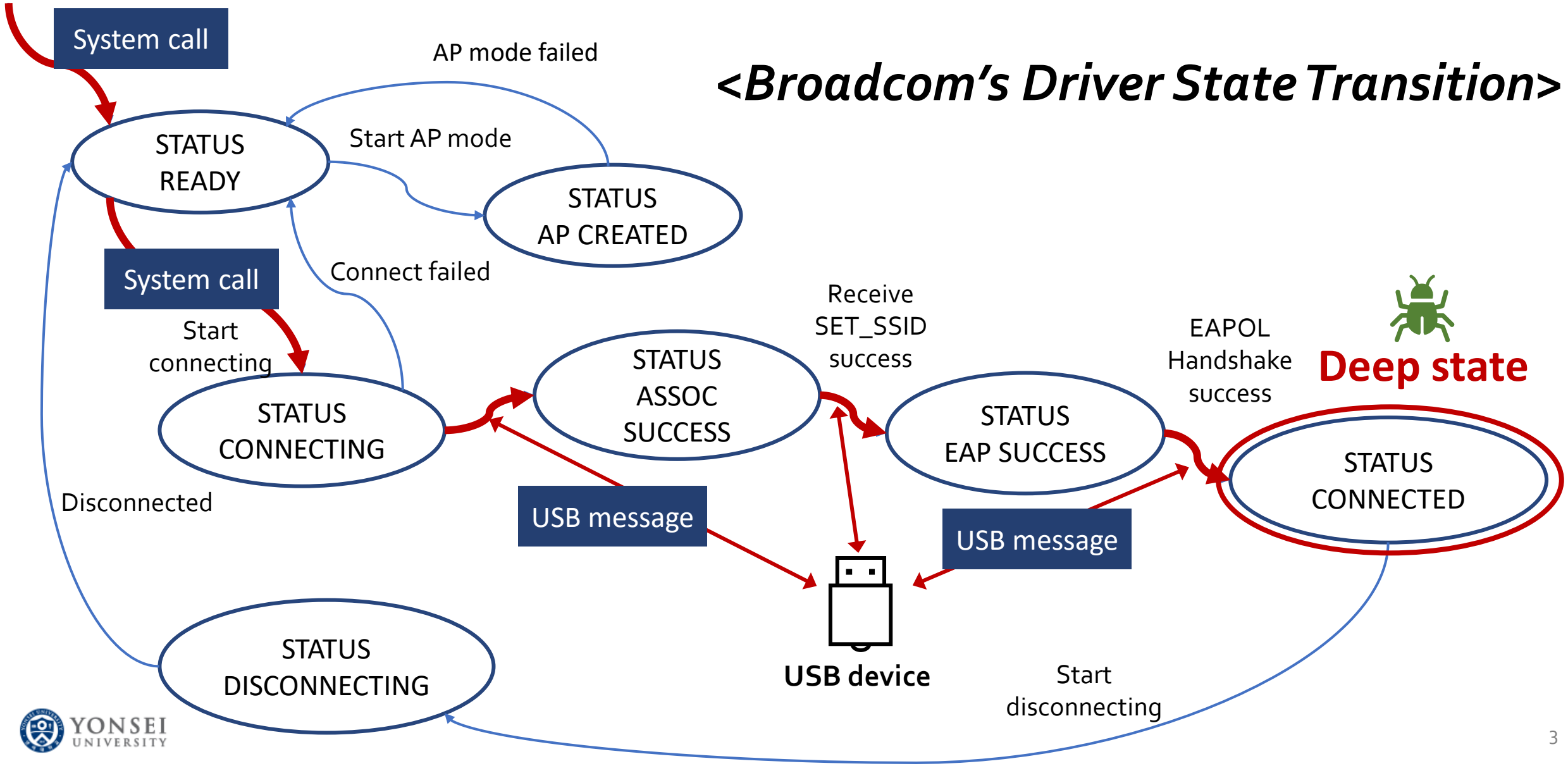
YONSEI UNIVERSITY

# Low Code Coverage of USB Device Drivers

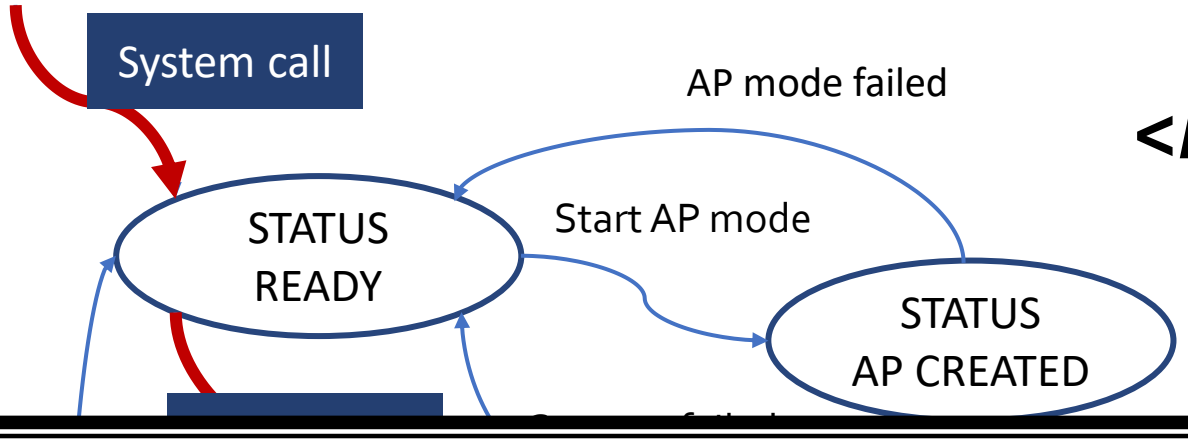
➢ **USB driver coverage of syzbot (as of July 2023)**

| Vendor | Driver source code | Total basic block of code | Fuzzing code coverage (%) |
|---|---|---|---|
| **Qualcomm** | drivers/net/wireless/ath | 5436 | **1%** |
| **Broadcom** | drivers/net/wireless/broadcom | 27881 | **2%** |
| | drivers/bluetooth/btbcm.c | 212 | **0%** |
| **Mediatek** | drivers/net/wireless/mediatek | 2190 | **0%** |
| **Ralink** | drivers/net/wireless/ralink | 4034 | **0%** |
| **Realtek** | drivers/net/wireless/realtek | 30250 | **1%** |
| **CSR** | drivers/bluetooth/btusb.c | 980 | **11%** |
| **NXP** | drivers/nfc/pn533 | 664 | **11%** |

# Statefulness of USB Device Drivers

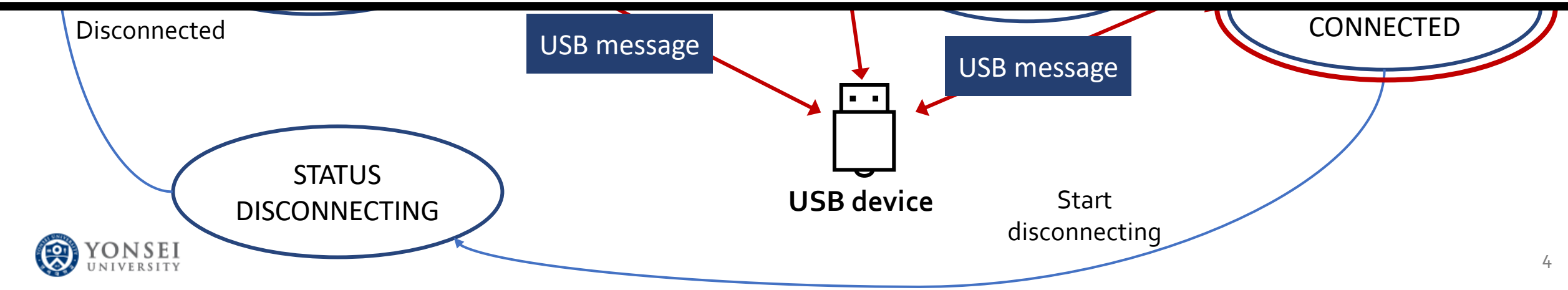


# Statefulness of USB Device Drivers



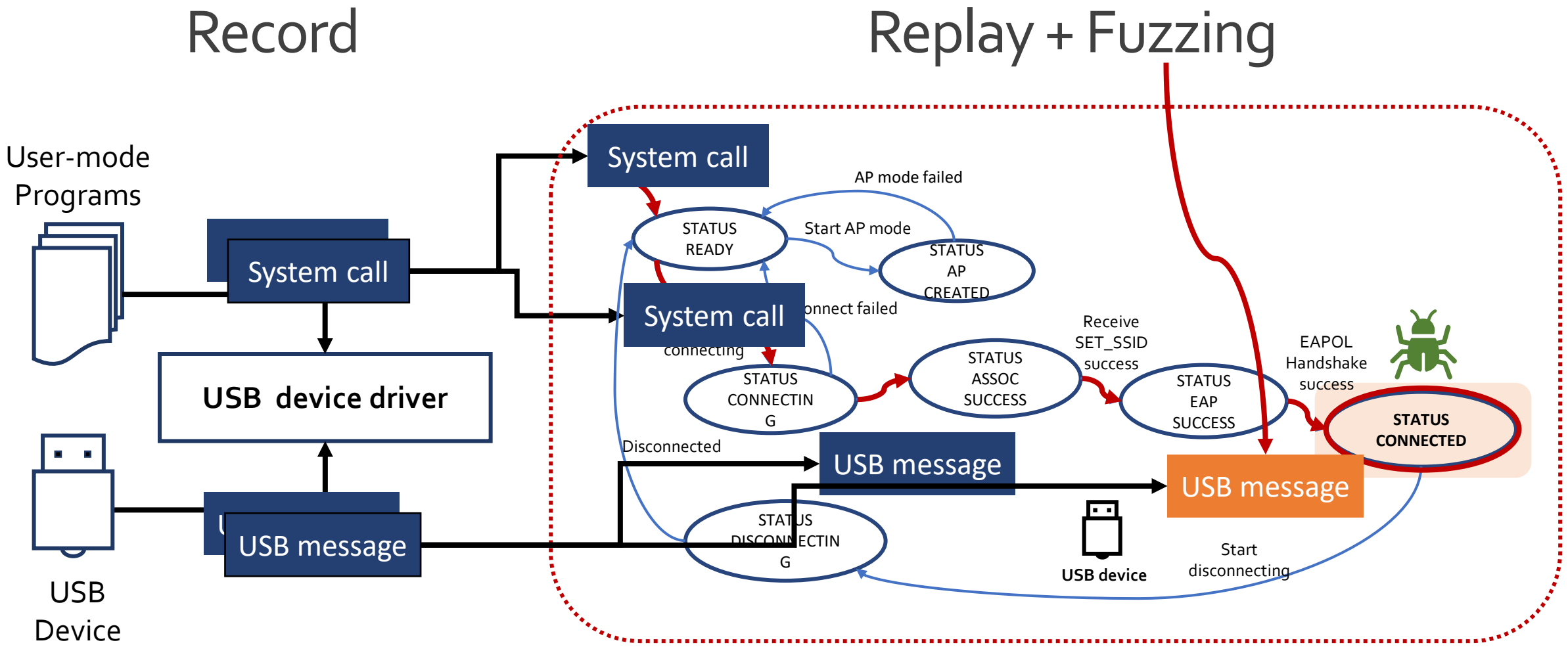

***<Broadcom's Driver State Transition>***

***A long sequence of specific inputs are required.***




YONSEI UNIVERSITY

# Our Approach: Combining Record-and-Replay with Fuzzing



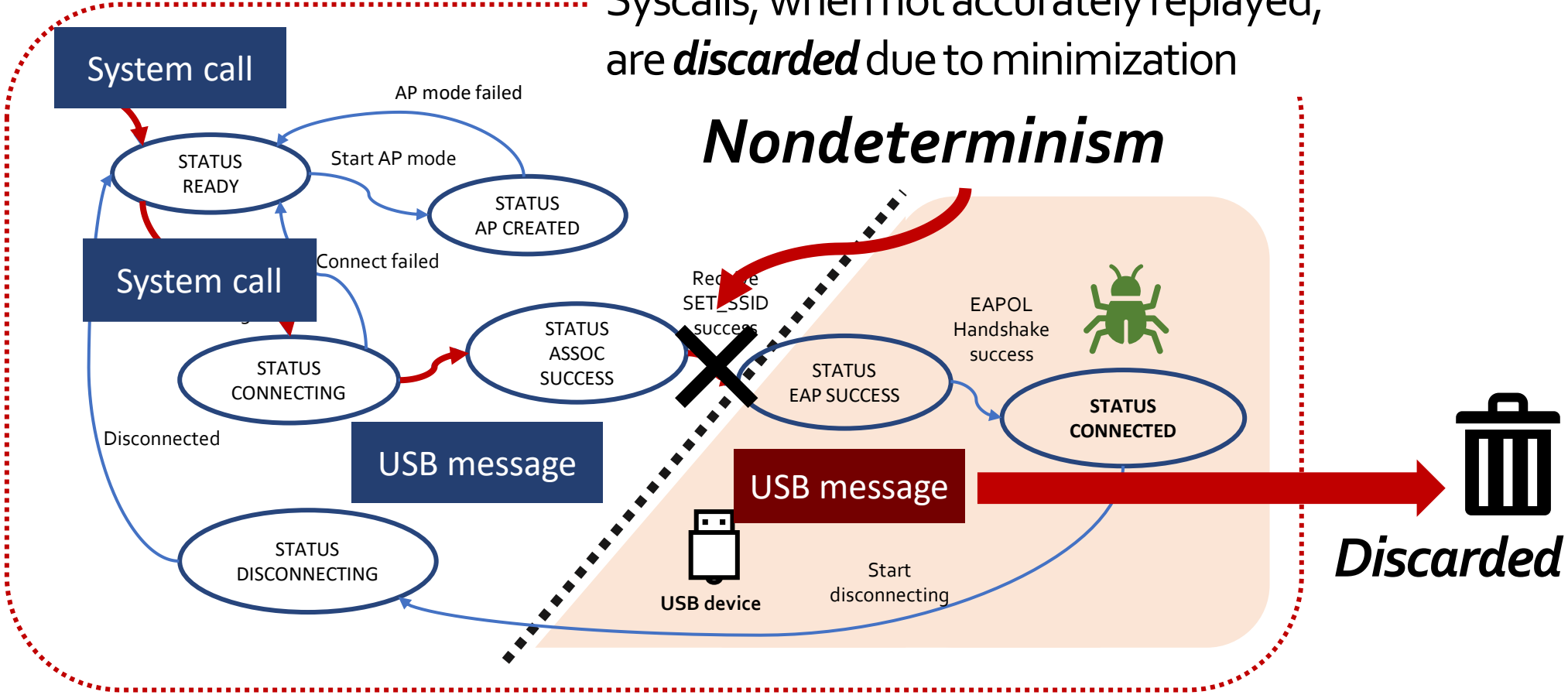
# Limitations of Coverage-Guided Evolutionary Fuzzing

**State-of-the-art evolves the corpus towards smaller inputs based on coverage:**

- **Syzkaller's fuzzing algorithm**
- **USBFuzz (uses AFL)'s fuzzing algorithm**

## 1. Minimization:

Syscalls, when not accurately replayed, are ***discarded*** due to minimization



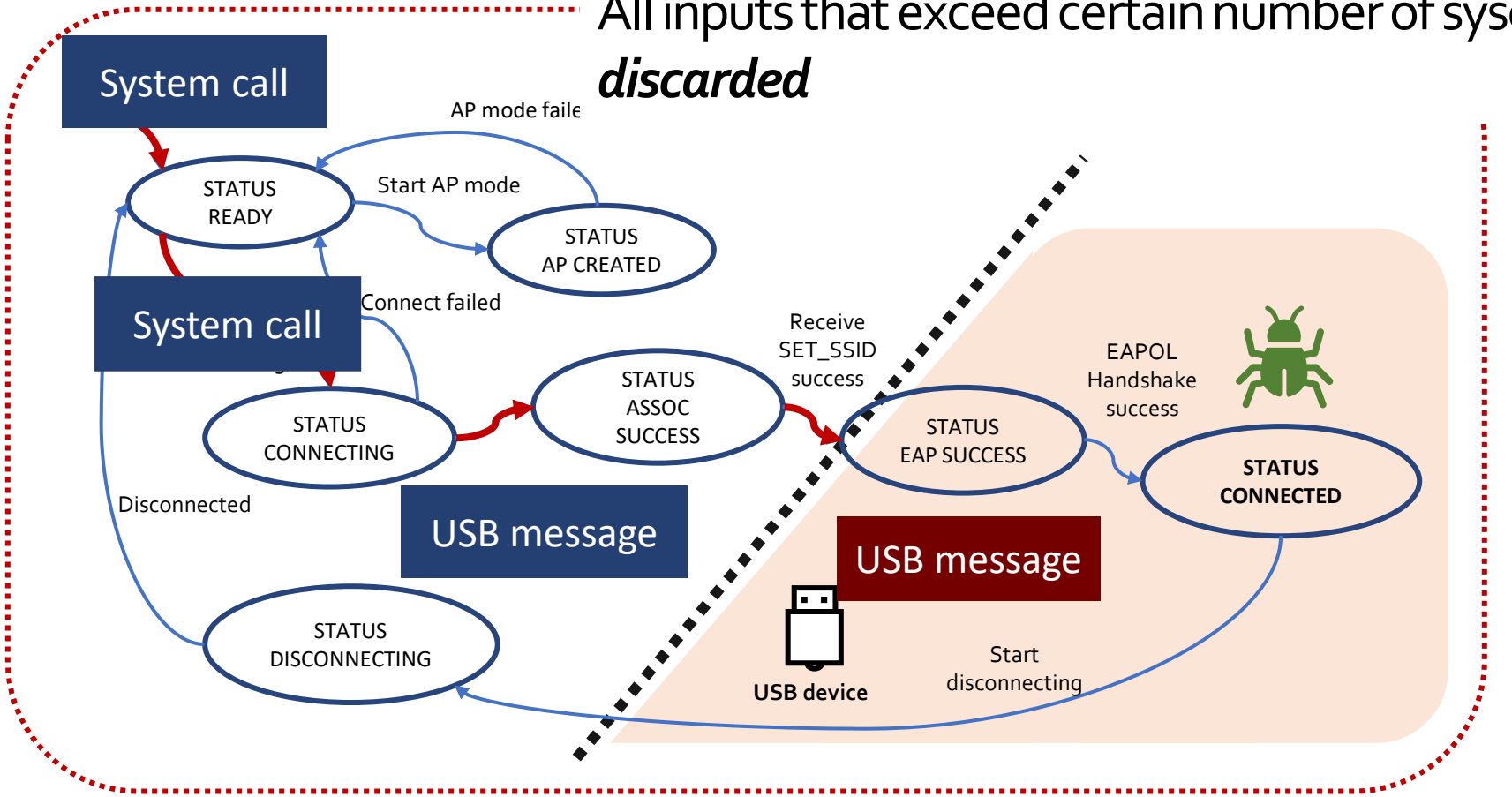
# Limitations of Coverage-Guided Evolutionary Fuzzing

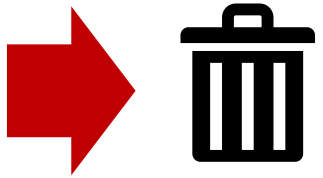
**State-of-the-art evolves the corpus towards smaller inputs based on coverage:**

- Syzkaller's fuzzing algorithm
- USBFuzz (uses AFL)'s fuzzing algorithm

## 2. Hard cap on the number of syscalls:

All inputs that exceed certain number of syscalls are ***discarded***



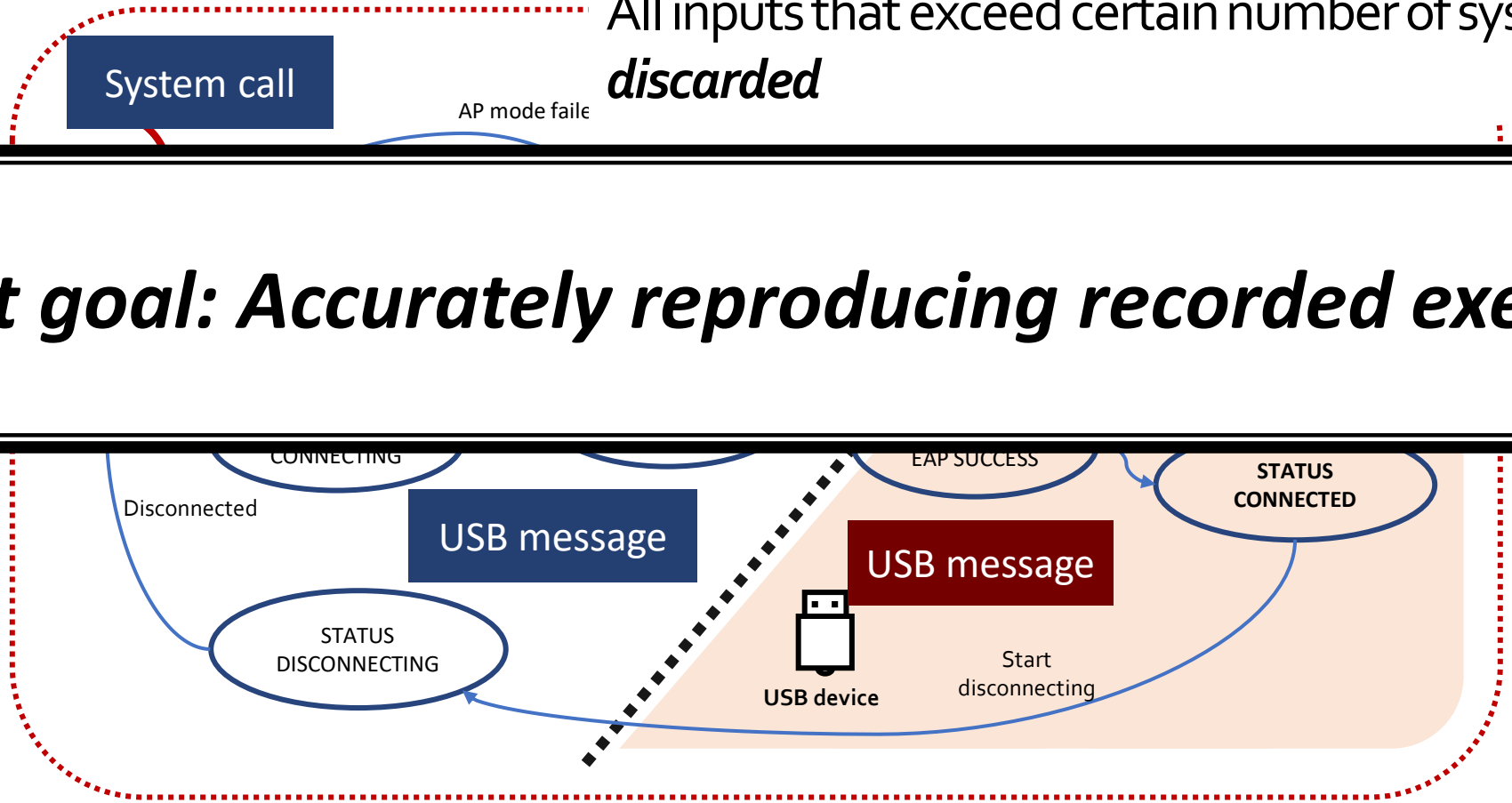
# Limitations of Coverage-Guided Evolutionary Fuzzing

**State-of-the-art evolves the corpus towards smaller inputs based on coverage:**

- Syzkaller's fuzzing algorithm
- USBFuzz (uses AFL)'s fuzzing algorithm

**2. Hard cap on the number of syscalls:**

All inputs that exceed certain number of syscalls are ***discarded***



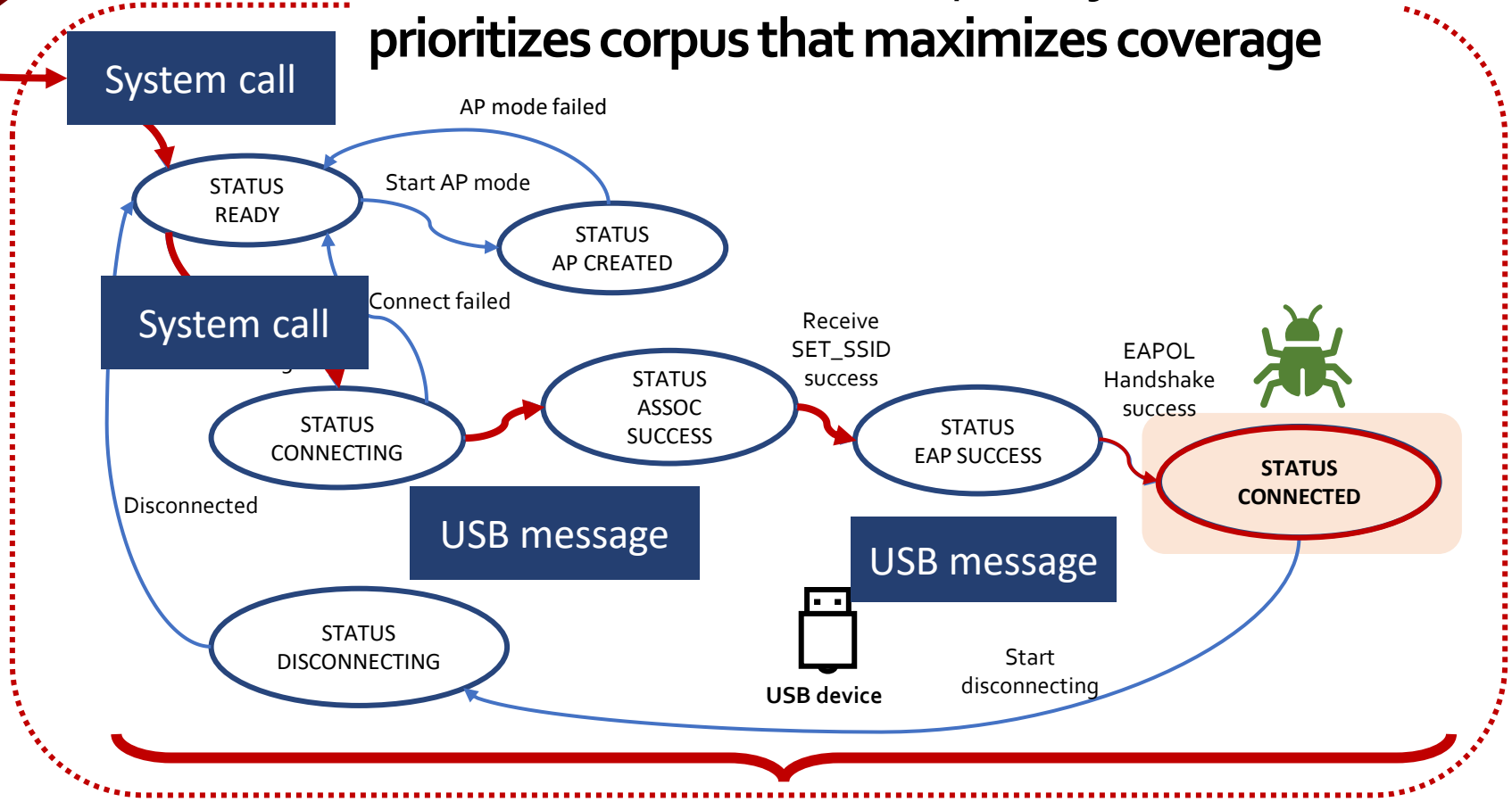

***Our first goal: Accurately reproducing recorded executions***

# Limitations of Coverage-Guided Evolutionary Fuzzing

System call

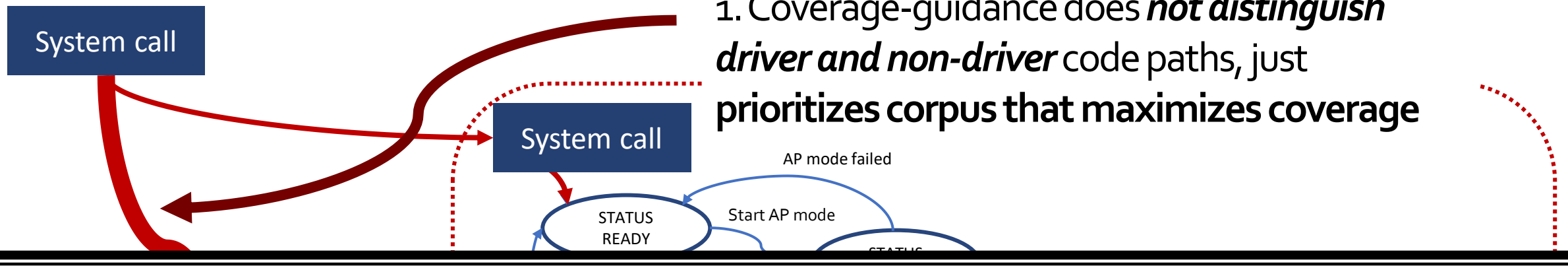
Non-driver kernel code

1. Coverage-guidance does ***not distinguish driver and non-driver*** code paths, just **prioritizes corpus that maximizes coverage**



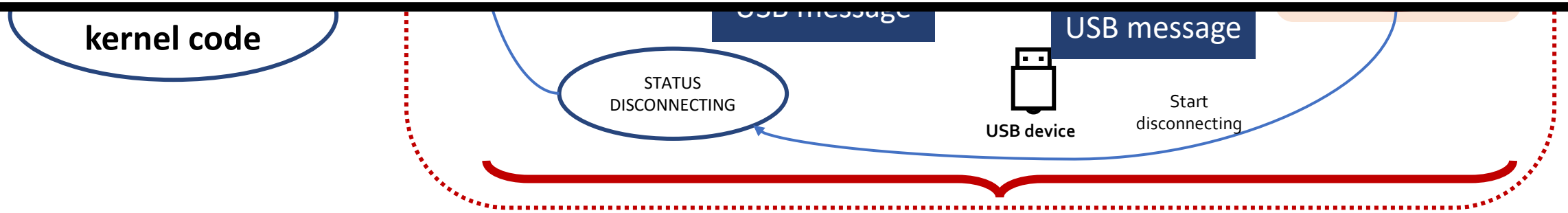

**2.Lengthy inputs (exceeding thousands) and long execution time (5~20 seconds)** make fuzzing extremely slow

# Limitations of Coverage-Guided Evolutionary Fuzzing




1. Coverage-guidance does ***not distinguish driver and non-driver*** code paths, just **prioritizes corpus that maximizes coverage**

***Our second goal: Efficient fuzzing for deep bugs in drivers***




**2.Lengthy inputs (exceeding thousands) and long execution time (5~20 seconds)** make fuzzing extremely slow

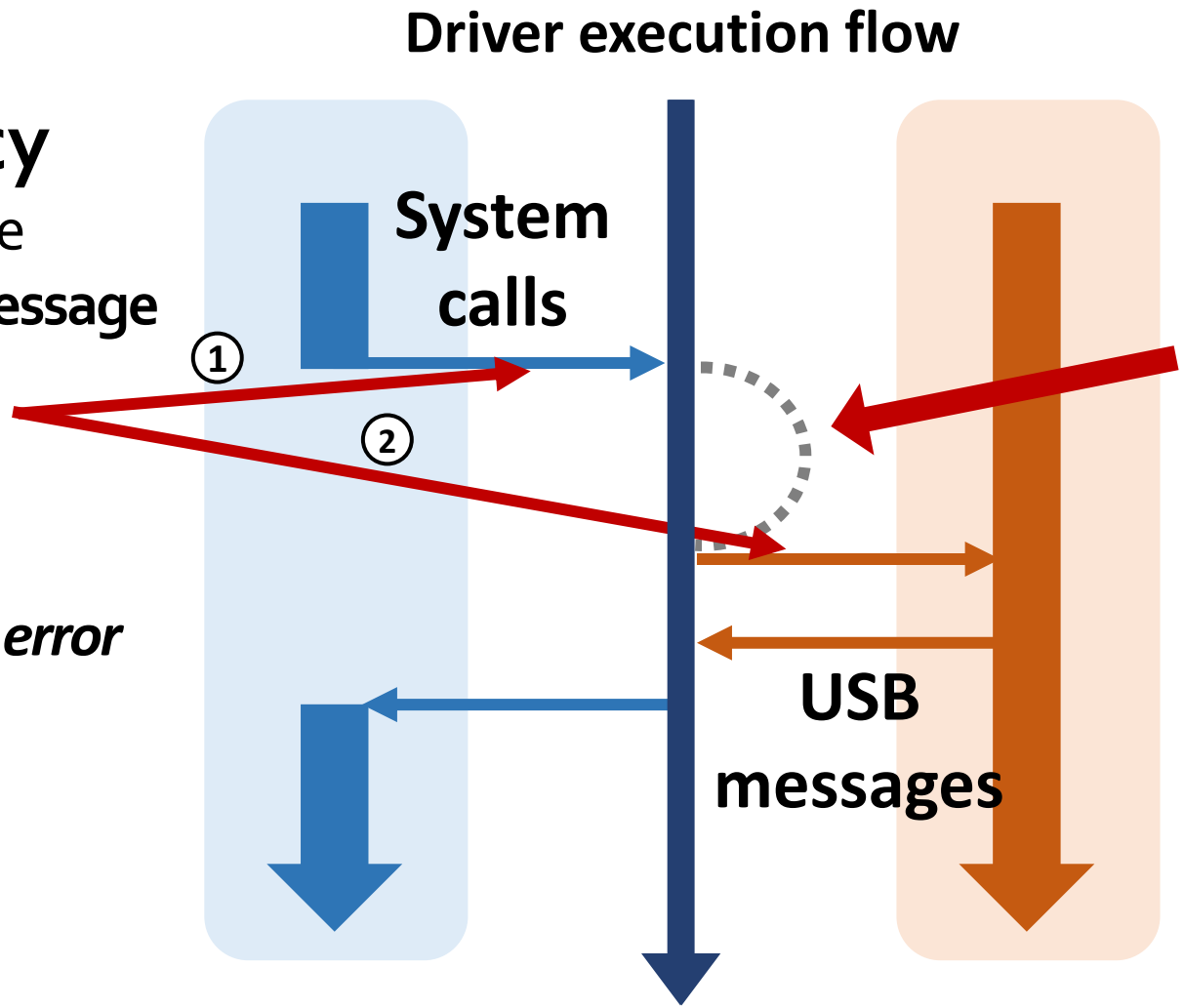
# Challenges of Accurate Replay: Sensitive to Timing of Input Injection

## 1. Concurrency

System calls trigger the **generation of USB message requests** by the driver (e.g., blocking I/O)

➡

***USB response timeout error***



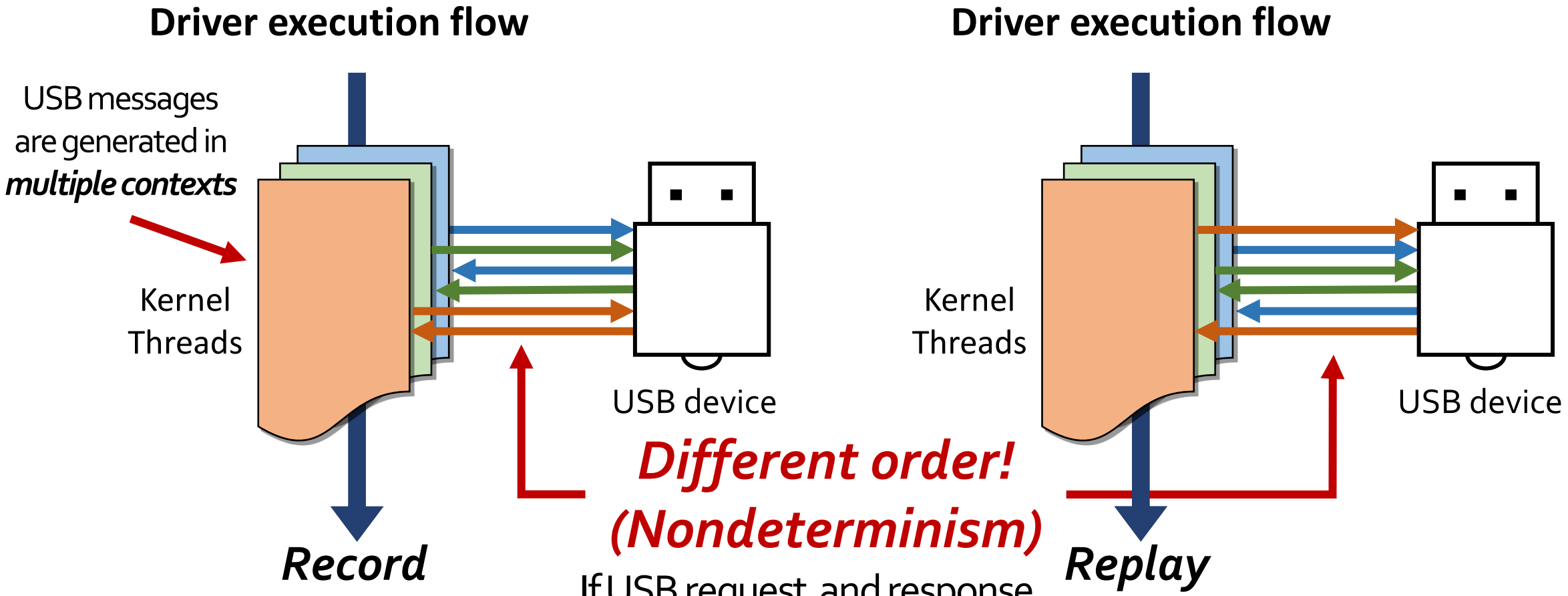

## 2. Delay

Driver requires ***a certain delay*** to generate USB messages. ( e.g., delay queue … )

➡

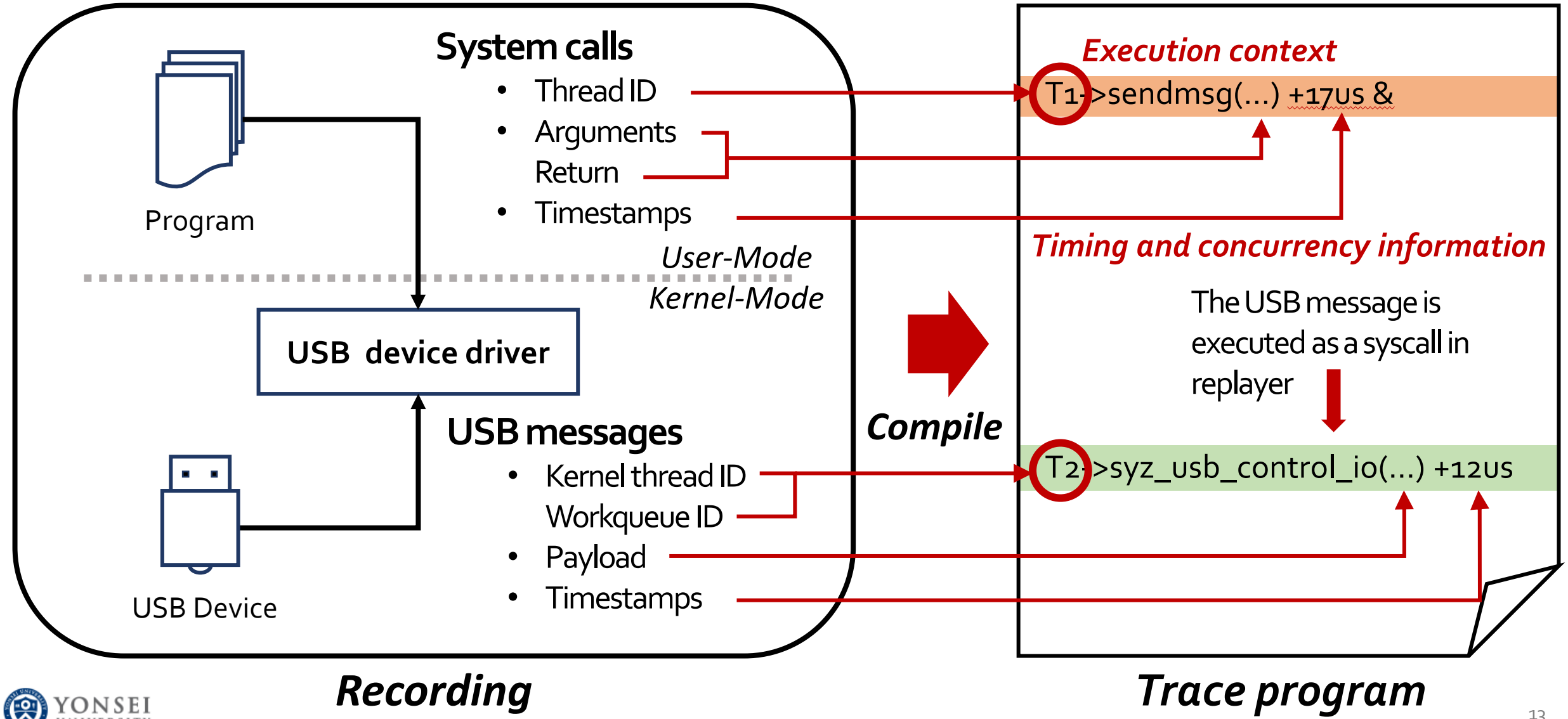
respond to USB messages ***without the corresponding USB request***

# Challenges of Accurate Replay: Unordered Concurrent USB Requests

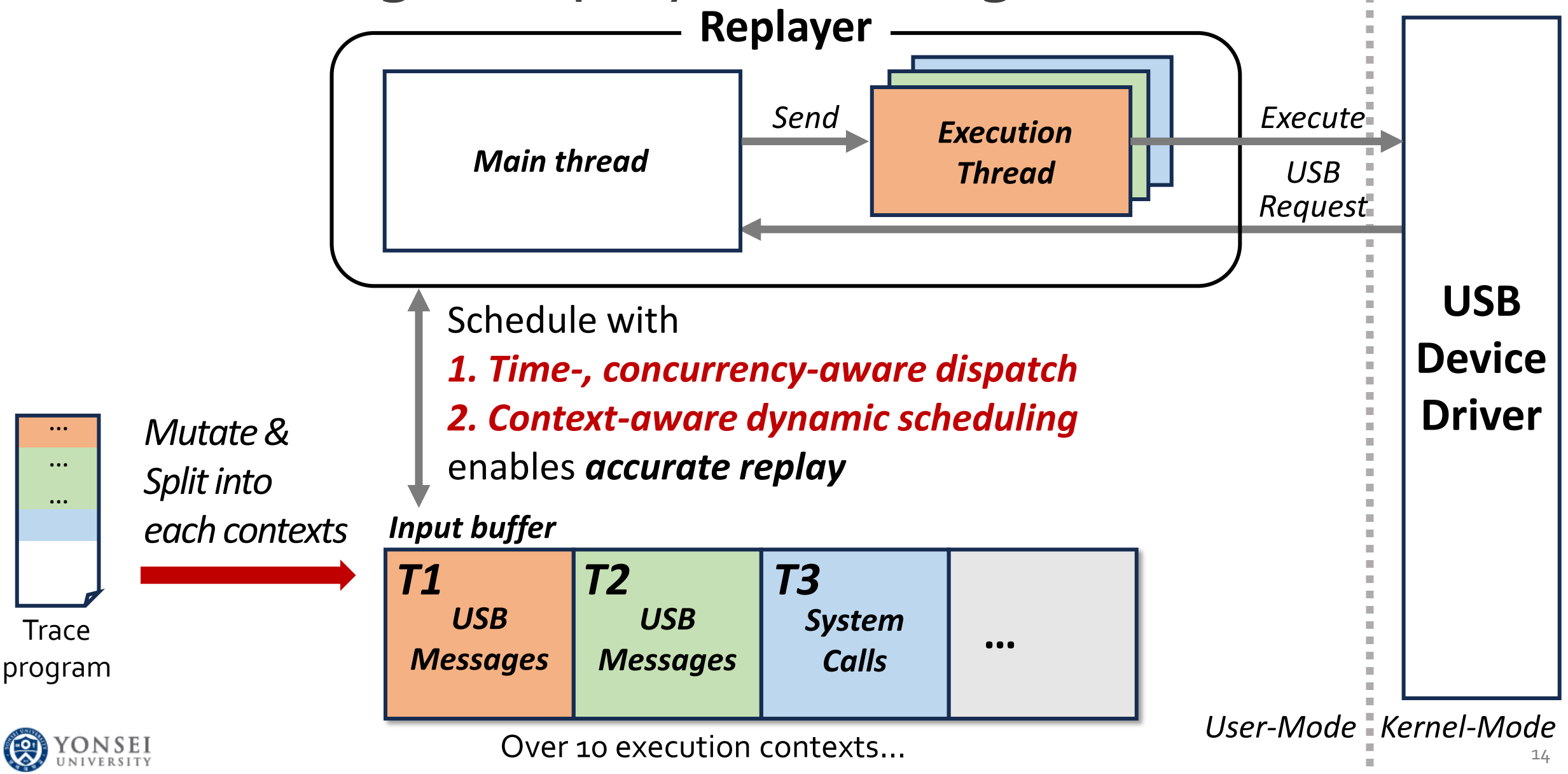



➡ Driver may execute ***differently from the recording***, resulting in errors

# ReUSB Design: Recording (Phase 1)



# ReUSB Design: Replay & Fuzzing (Phase 2)



# Time-, Concurrency-Aware Dispatch: Concurrency

T1->ioctl(…) + 412 us &
T3->syz_usb_control_io(…) + 2066 us
T1->poll(…) + 51773us
T3->syz_usb_control_io(…) + 131 us

② ioctl() triggers the generation of USB request and is ***blocked***

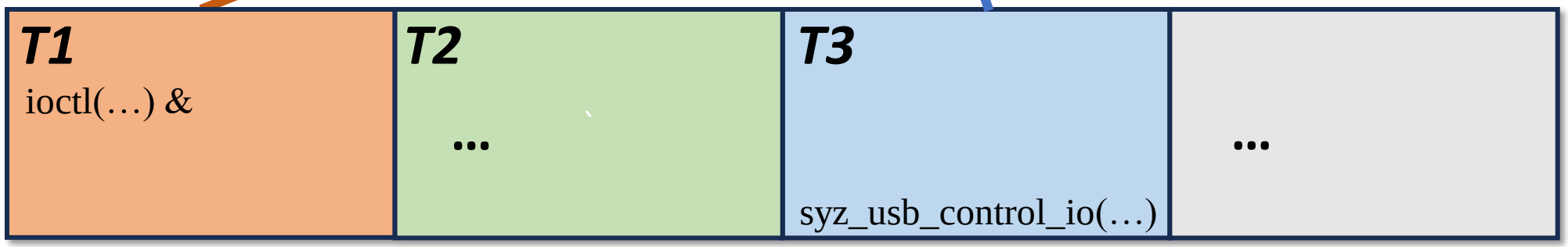
***Replayer***

***Execute***

**USB Device Driver**

① Schedule & execute ***blocking I/O*** ioctl()

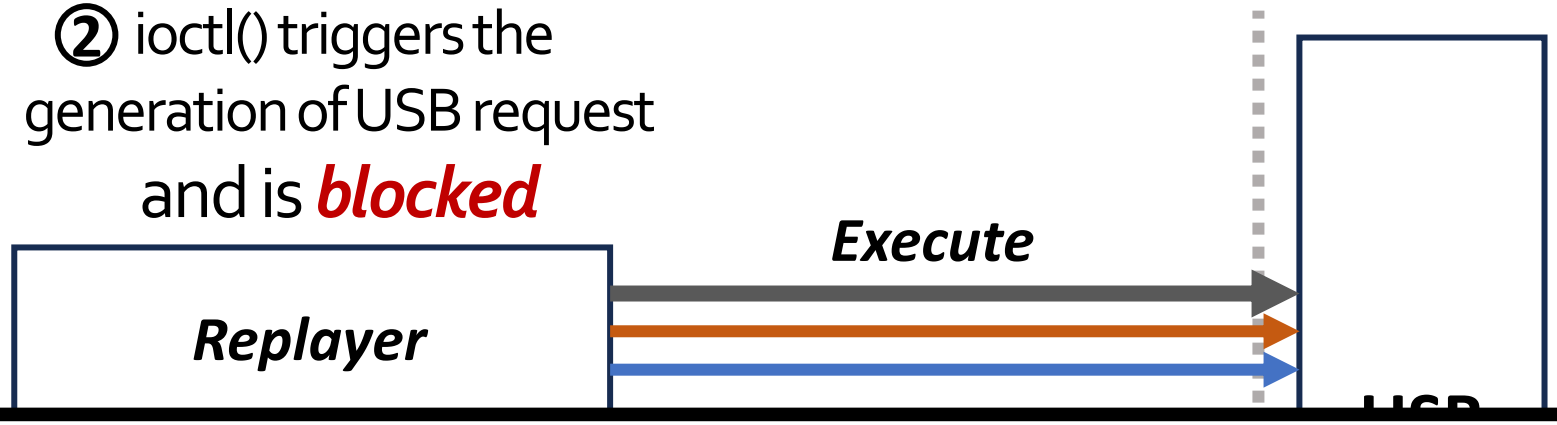
③ Schedule & execute USB response message ***without waiting for return***



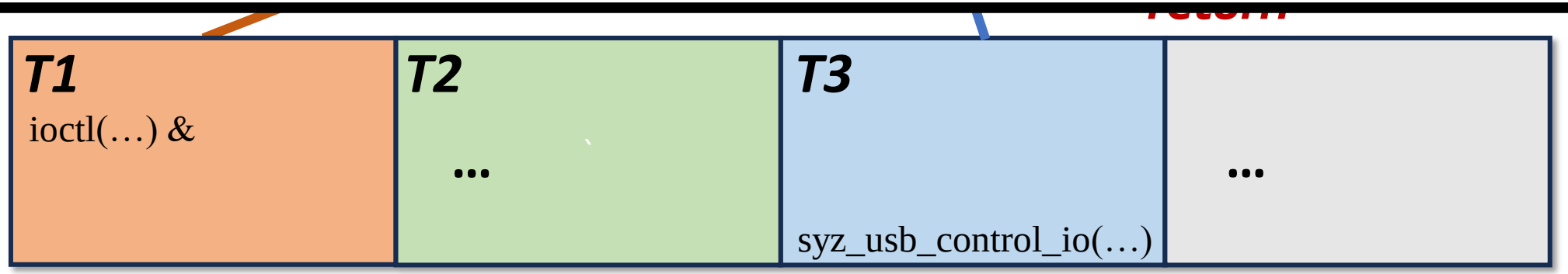
# Time-, Concurrency-Aware Dispatch: Concurrency

T1->ioctl(…) + 412 us &
T3->syz_usb_control_io(…) + 2066 us
T1->poll(…) + 51773us
T3->syz_usb_control_io(…) + 131 us

② ioctl() triggers the generation of USB request and is ***blocked***




***ReUSB can execute system calls in different contexts asynchronously.***



# Time-, Concurrency-Aware Dispatch: Delay

T1->ioctl(…) + 412 us &
T3->syz_usb_control_io(…) + 2066 us
T1->poll(…) + 51773us
T3->syz_usb_control_io(…) + 131 us

② The replayer experiences a ***temporary delay***

***Replayer***

***Execute***

**USB Device Driver**

① Schedule, send, execute, and return from poll()

③ Schedule & execute next system call ***after delay***

| ***T1*** | ***T2*** | ***T3*** | |
|---|---|---|---|
| poll(…) +51773us | … | syz_usb_control_io(…) | … |

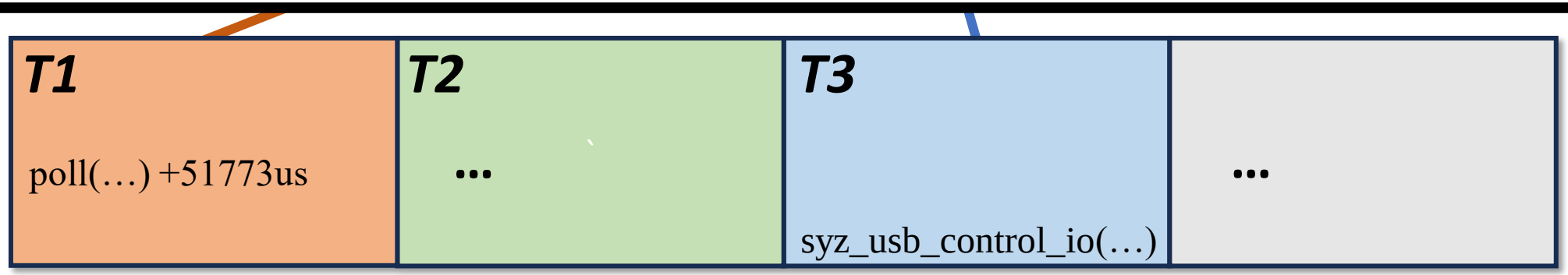
# Time-, Concurrency-Aware Dispatch: Delay

T1->ioctl(…) + 412 us &
T3->syz_usb_control_io(…) + 2066 us
T1->poll(…) + 51773us
T3->syz_usb_control_io(…) + 131 us

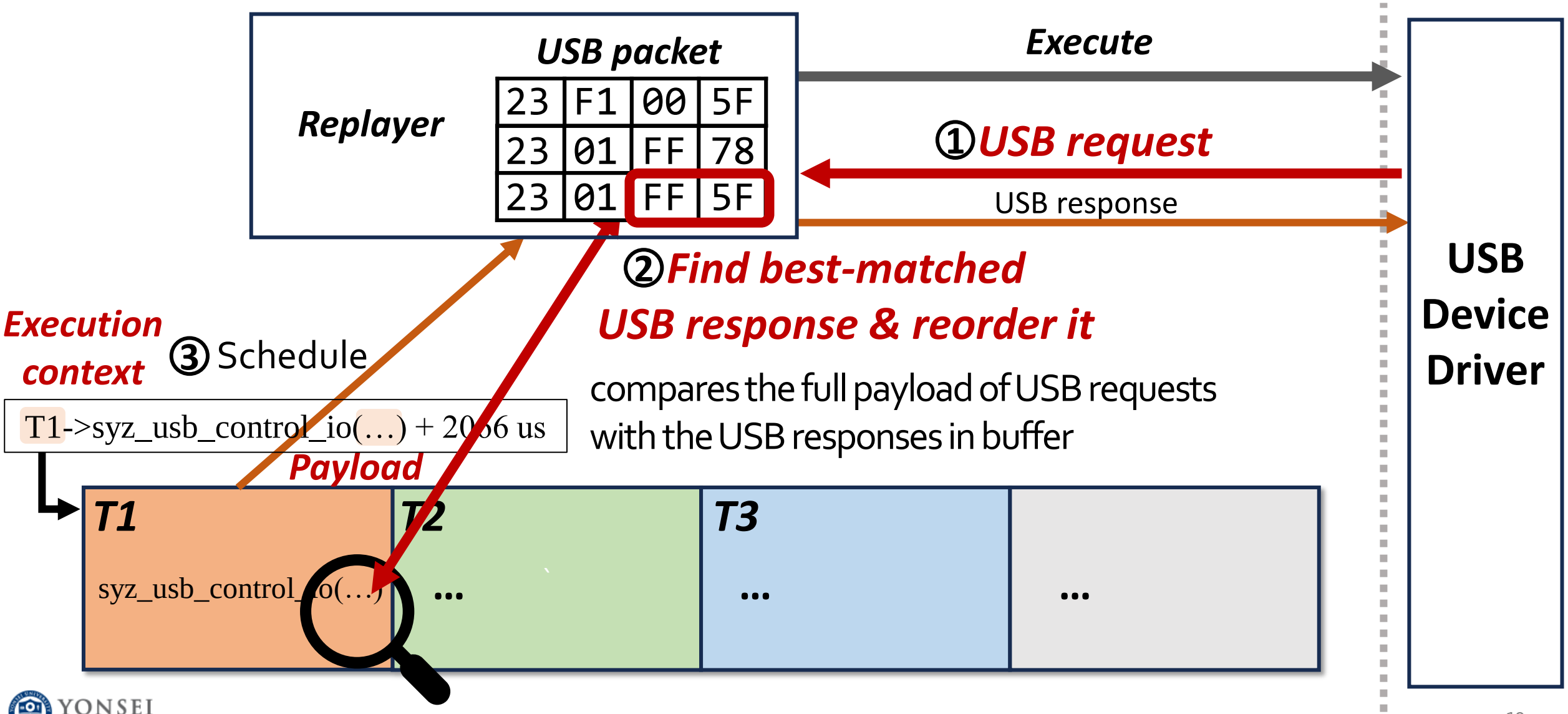
② The replayer experiences a ***temporary delay***



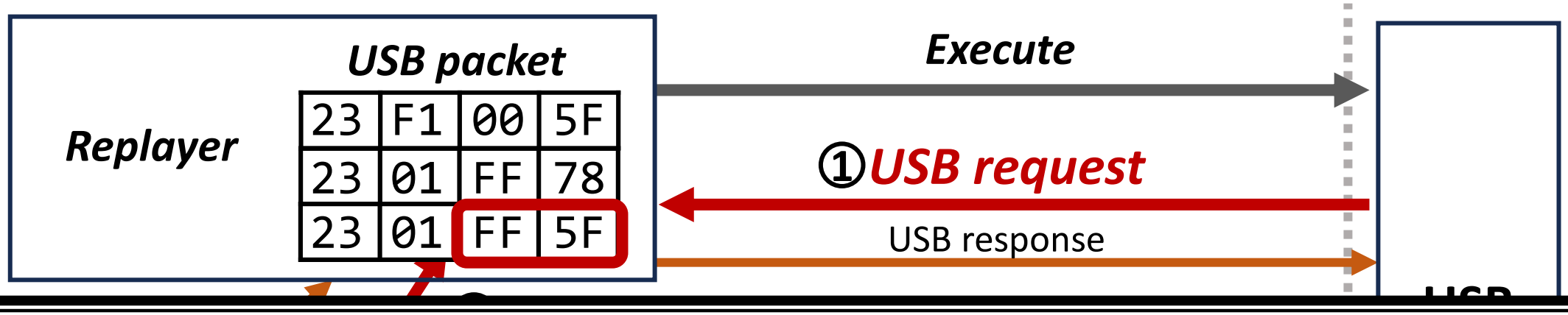

***ReUSB can inject delays between system calls to account for time-dependent behavior of drivers.***



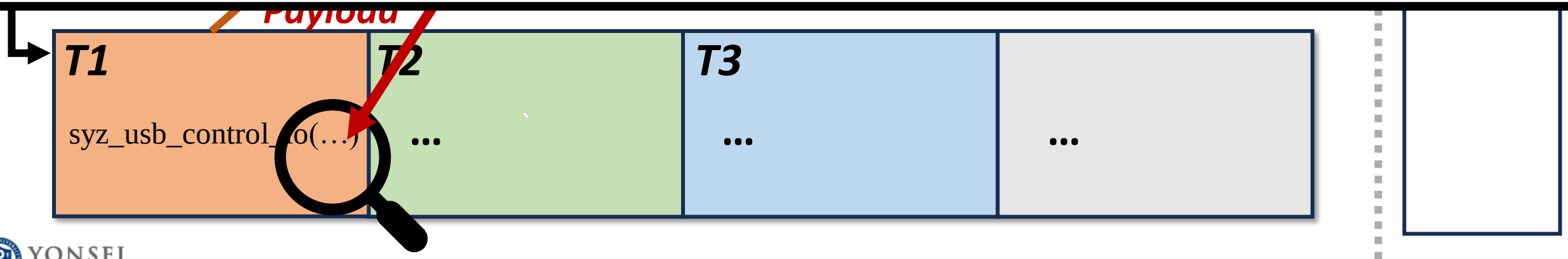
# Context-Aware Dynamic Scheduling



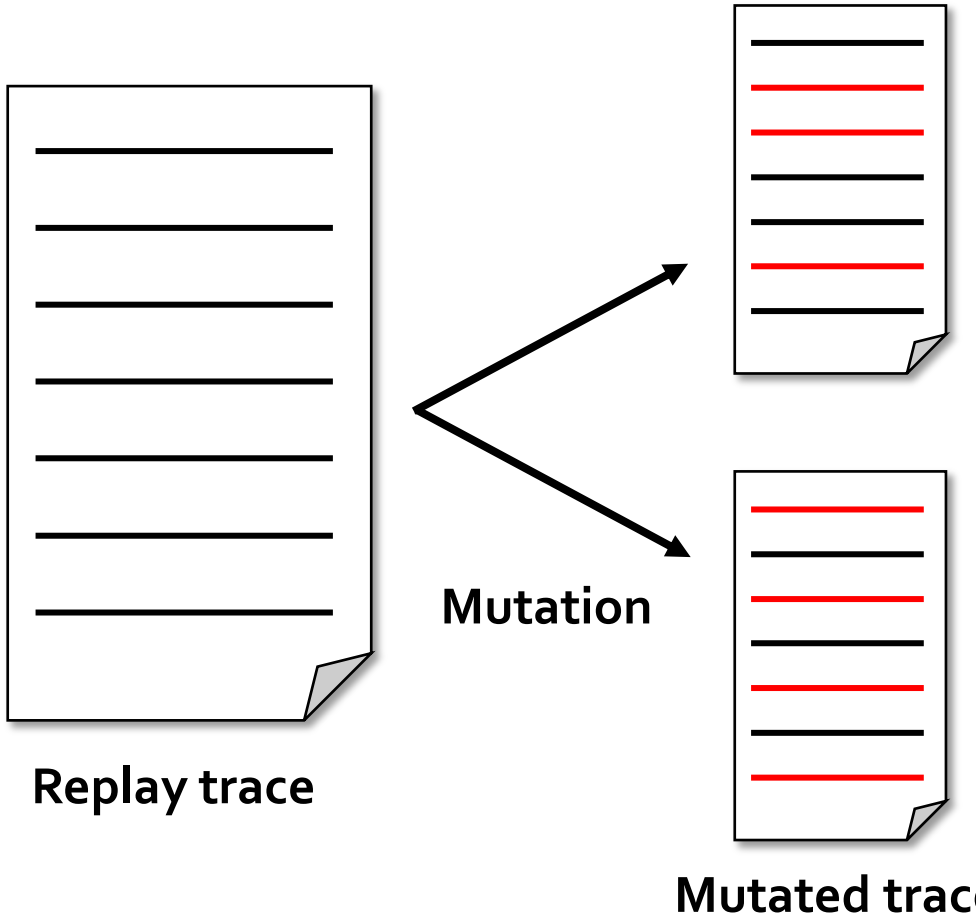
# Context-Aware Dynamic Scheduling

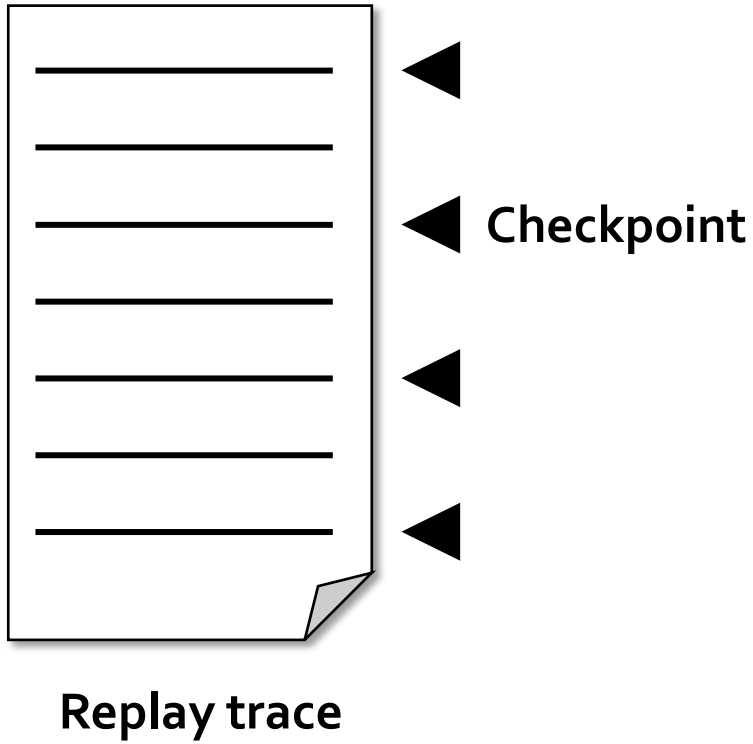



***ReUSB can handle unordered concurrent USB requests by reordering USB response injection.***



# Efficient USB Driver Fuzzing




Too large mutation surface

➡ ***Replay-guided fuzzing***

Excessively slow and lengthy

➡ ***Replay checkpointing***

# Efficient USB Driver Fuzzing


***More information available in the paper about our mutation & replay checkpointing policies!***


Too large mutation surface
➡ ***Replay-guided fuzzing***

Excessively slow and lengthy
➡ ***Replay checkpointing***

# Implementation

- Qemu 4.0, Linux KVM
- XHCI USB controller
- STRACE, Wireshark, and USBMON
- Syzkaller
- Agamotto
- Linux's raw gadget

# 10 Wireless USB Device Drivers

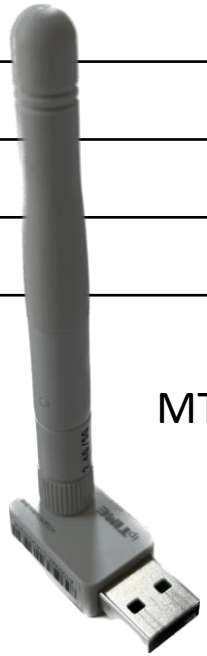
* Out-of-tree drivers whose source code is available at the shown URL

| Class | Vendor | Device | Driver Source Code |
|---|---|---|---|
| Wi-Fi | Broadcom | BCM43236 | drivers/net/wireless/broadcom/brcm80211 |
| | Qualcomm | AR9271 | drivers/net/wireless/ath/ath9k |
| | Ralink | RT5370 | drivers/net/wireless/ralink/rt2x00 |
| | Realtek | RTL8821BU<br>RTL8821AU | github.com/morrownr/88x2bu-20210702*<br>github.com/aircrack-ng/rtl8812au* |
| | Mediatek | MT7601U<br>MT7610U | drivers/net/wireless/mediatek/mt7601u<br>drivers/net/wireless/mediatek/mt76/mt76x0 |
| Blue-tooth | Broadcom | BCM20702 | drivers/bluetooth/btbcm.c |
| | CSR | CSR8510 | drivers/bluetooth/btusb.c |
| NFC | NXP | PN533 | drivers/nfc/pn533 |

BCM43236

CSR8510

PN533



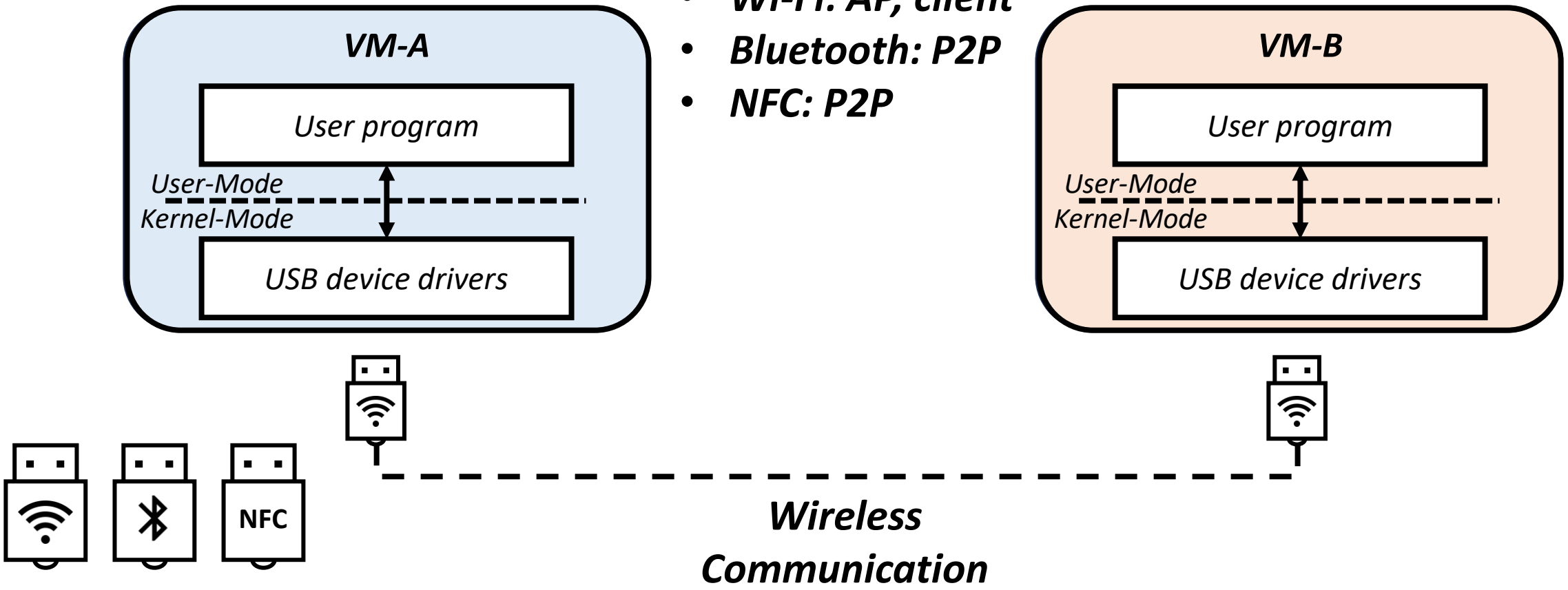
MT7601U

YONSEI UNIVERSITY

# Dual-VM Recording Environment

***Dual-VM recording of typical communication scenarios***

- ***Wi-Fi: AP, client***
- ***Bluetooth: P2P***
- ***NFC: P2P***



# Result: Number of Input Injections

| | VM-A | | VM-B | | |
|---|---|---|---|---|---|
| | Device | # of action (Syscall/USB) | Device | # of action (Syscall/USB) | Duration (sec.) |
| Wi-FI | BCM43236 | **1,894**(1,495/ 399) | MT7601U | - | 20 |
| | AR9271 | **8,143**(1,577/ 6,566) | MT7601U | - | 19 |
| | RT5370 | **6,311**(1,568/ 4,743) | MT7601U | - | 19 |
| | RTL8812BU | **24,529**(2,761/21,768) | MT7601U | - | 23 |
| | RTL8821AU | **9,328**(2,550/ 6,778) | MT7601U | - | 23 |
| | MT7601U | **4,099**(1,494/ 2,605) | MT7601U | **9,489**(2,047/ 7,442) | 20 |
| | MU7610U | **15,011**(2,639/12,372) | BCM43236 | **1,484**(1,051/ 433) | 20 |
| | MT7601U | - | AR9271 | **11,094**(2,600/ 8,494) | 24 |
| | MT7601U | - | RT5370 | **11,272**(2,244/ 9,028) | 21 |
| | MT7601U | - | RTL8812BU | **12,577**(1,104/11,473) | 21 |
| | MT7601U | - | RTL8821AU | **6,728**(1,104/ 5,624) | 19 |
| Bluetooth | BCM20702 | **6,108**(3,866/ 2,242) | CSR8510 | **1,219**(1,037/ 182) | 21 |
| | CSR8510 | **9,219**(6,423/ 2,796) | BCM20702 | **2,004**(1,035/ 969) | 24 |
| NFC | PN533 | 475( 437/ 38) | PN533 | 528( 484/ 44) | 4 |

**The number of recorded input mostly exceeds thousand!**

YONSEI UNIVERSITY

# Result: Coverage Increase in Replay

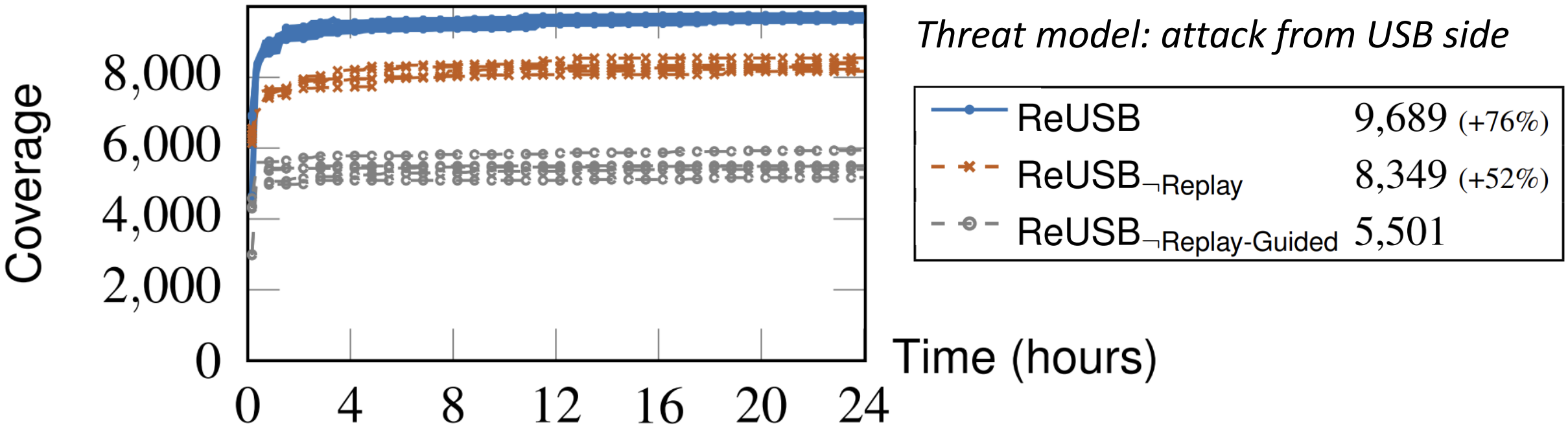
| | | Record | Replay | | |
|---|---|---|---|---|---|
| | | | Baseline | After Time-and-concurrency -aware dispatch | After Context-aware dynamic scheduling |
| AR9271 | Client<br>AP | 3,956<br>4,311 | 1,709 (34.5%)<br>1,447 (33.6%) | **2,808 (56.7%)**<br>**2,721 (63.1%)** | 2,808 (56.7%)<br>2,721 (63.1%) |
| BCM43236 | Client<br>AP | 3,533<br>3,205 | 1,346 (38.1%)<br>1,295 (40.4%) | **2,507 (71.0%)**<br>**1,440 (44.9%)** | **3,441 (97.4%)**<br>**2,827 (88.2%)** |
| MT7610U | Client<br>AP | 4,458<br>3,976 | 1,629 (36.5%)<br>1,326 (33.4%) | **1,762 (39.5%)**<br>**1,437 (36.1%)** | **2,265 (50.8%)**<br>**2,240 (56.3%)** |
| RT5370 | Client<br>AP | 4,232<br>3,724 | 1,269 (30.0%)<br>1,005 (27.0%) | **1,832 (43.3%)**<br>**2,073 (55.7%)** | **1,933 (45.7%)**<br>**2,831 (76.0%)** |
| | | … | … | … | … |
| Geometric mean | | | 42.0% | **62.5%** | **69.7%** |

# Result: Finding 15 Previously Unknown Bugs

| Device | Role(s) | Error Type | Upstream patch | |
|---|---|---|---|---|
| BCM43236 | Client & AP | Slab-out-of -bounds | 4920ab1 | |
| | Client & AP | Slab-out-of -bounds | 4920ab1 | |
| | Client & AP | Stack-out-of –bounds | 0a06cad | |
| | Client & AP | Stack-out-of -bounds | 660154d | |
| | Client & AP | Null pointer dereference | 683b972 | |
| | Client & AP | Shift-out-of-bounds | 81d17f6 | |
| | Client & AP | Slab-out-of -bounds | 6788ba8 | **CVE-2022-3628** |
| | Client | Slab-out-of -bounds | 0da40e0 | **CVE-2023-1380** |
| MT7601U | Client | Null pointer dereference | 803f317 | |
| MT7610U | Client & AP | Null pointer dereference | Bd5dac7 | |
| AR9271 | Client & AP | Stack-out-of -bounds | 8a2f35b | |
| | Client | Use-after-free | F099c5c | |
| PN533 | Master | Slab-out-of -bounds | 9f28157 | |
| | Master & Slave | Use-after-free | 9dab880 | |
| | Slave | Use-after-free | 4bb4db7 | |

YONSEI UNIVERSITY

# Result: Coverage Increase in Fuzzing

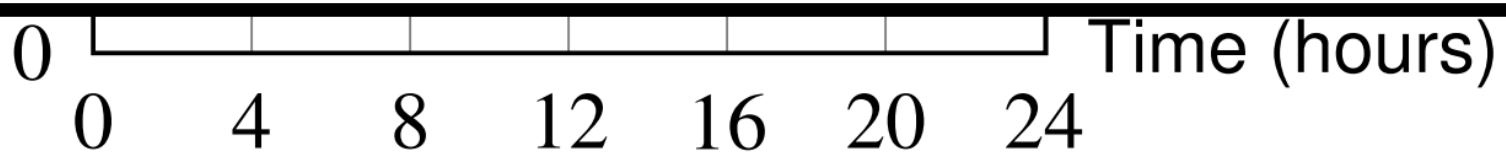



**Using all of our 19 trace programs as an initial seed corpus**

# Result: Coverage Increase in Fuzzing


***More evaluation results available in the paper!***

***(More coverage results, throughput, comparison with prior work, etc.)***




**Using all of our 19 trace programs as an initial seed corpus**

# Conclusion

- We proposed a replay-guided approach to USB driver fuzzing.
  - Controlling timing, concurrency, order of input injection matters for accurate replay of USB drivers.
  - The overhead stemming from accurate replay can be compensated through replay-guided fuzzing and replay checkpointing.

- We showed that replay-guided fuzzing is effective.
  - Fuzzed 10 stateful USB drivers, increased the coverage by up to 76%.
  - Found 20 bugs, of which **15** were previously unknown.
    - Obtained 2 CVEs: **CVE-2022-3628**, **CVE-2023-1380**

# Thank you!

Contact

Jisoo Jang, a Ph.D. student at Yonsei University

jisoo.jang@yonsei.ac.kr