μUSB: Practical and Safe USB Driver Reuse for Arm TrustZone

Xuankai Zhang[*], Sijin Li[*], Pei Meng[*], Meng Wang[†],
Yongzhao Zhang, Ting Chen, Xiaosong Zhang, and Liwei Guo
University of Electronic Science and Technology of China
[†]*CISPA Helmholtz Institute of Information Security*
* = co-primary authors



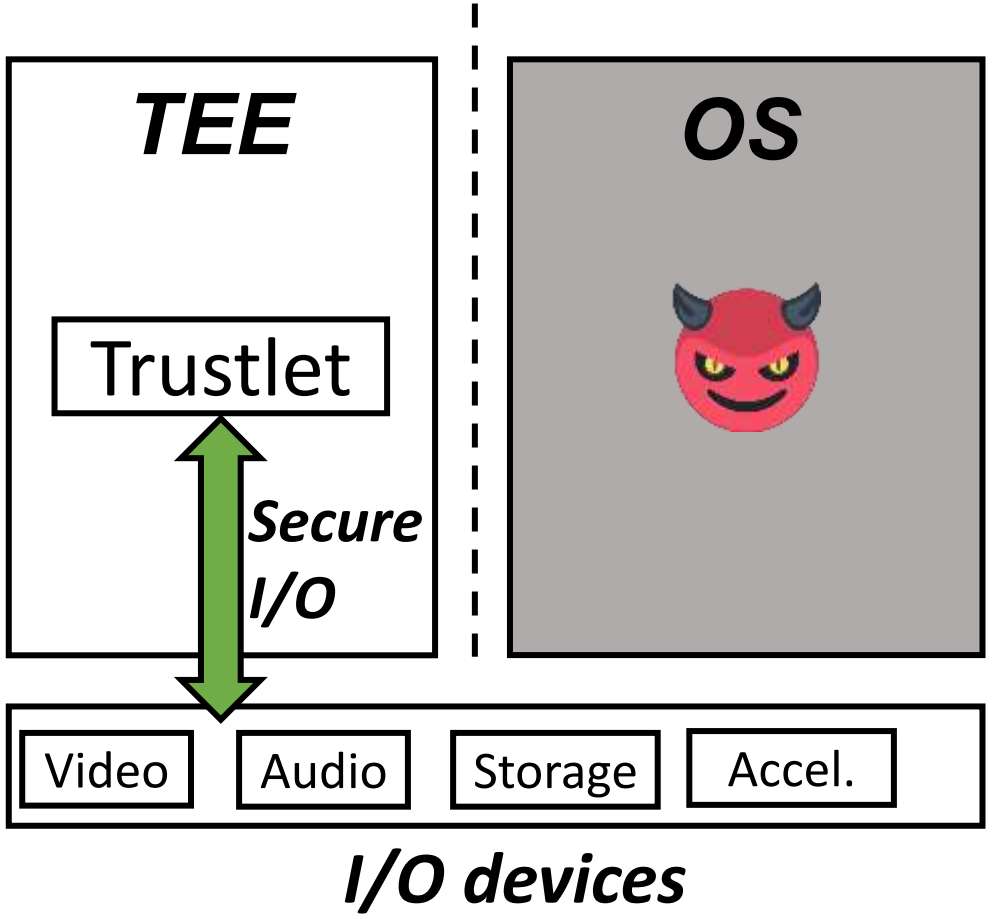


TEE & Secure I/O

- **Trusted Execution Environment (TEE)**
  - Isolates CPU executions of *trustlets*
  - Our target: Arm TrustZone (mobile/edge)
- **Secure I/O**
  - Isolates device accesses
- **Confidential data processing**
  - Keymaster, biometrics, Trusted UI…







The USB Secure I/O Dilemma

- **Important, yet missing USB support**
  - **Most diversely, widely deployed I/O devices**
    - a **WIDE** class: storage, HID, video, audio …
    - a **LARGE** variety: sensors of various types (e.g., vibration, light, …)
    - a **UNIVERSAL** interface: on all edge devices/SoCs
    - **NEW possibilities for confidential data processing**



Flash drive



Webcam



Card reader



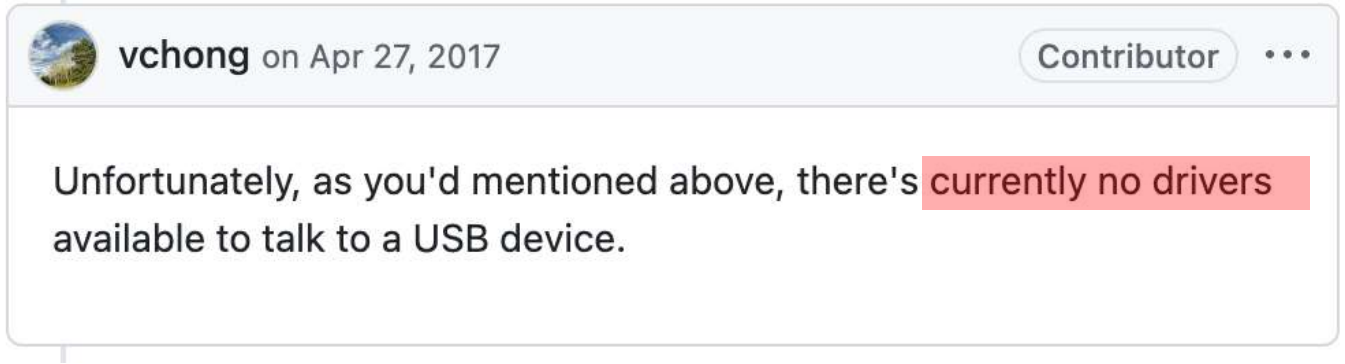
Env. sensor



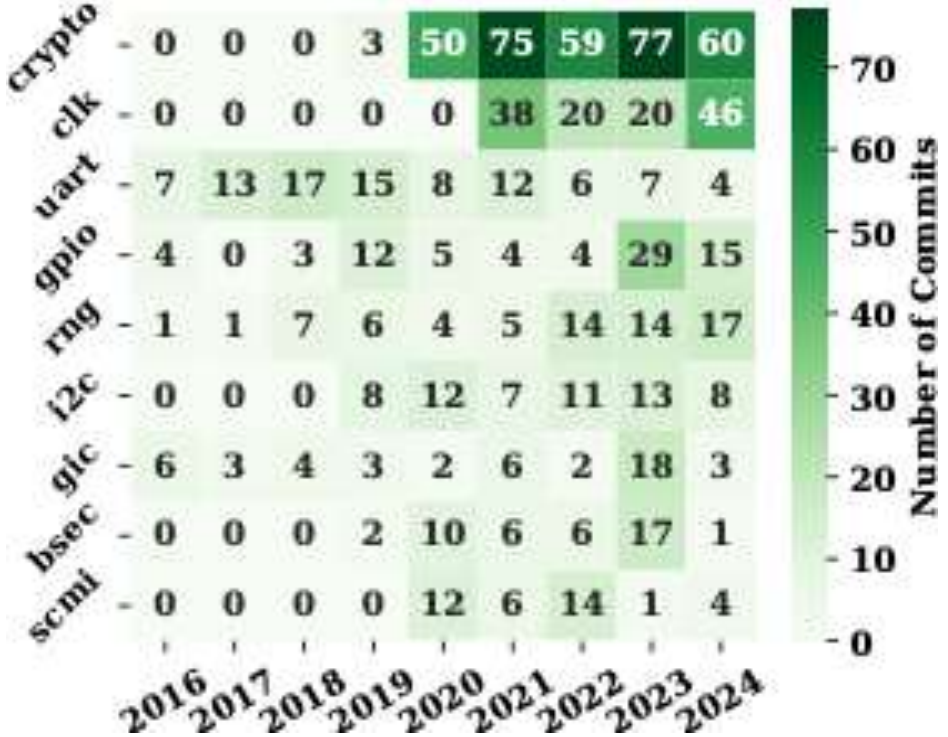
UWB sensor

The USB Secure I/O Dilemma (contd.)

- **Important, yet missing USB support**
  - **Absent from commodity TEEs**
    - First feature request: 2017
    - Recent driver commits only on simple I/O drivers
    - **NO sign of future development**



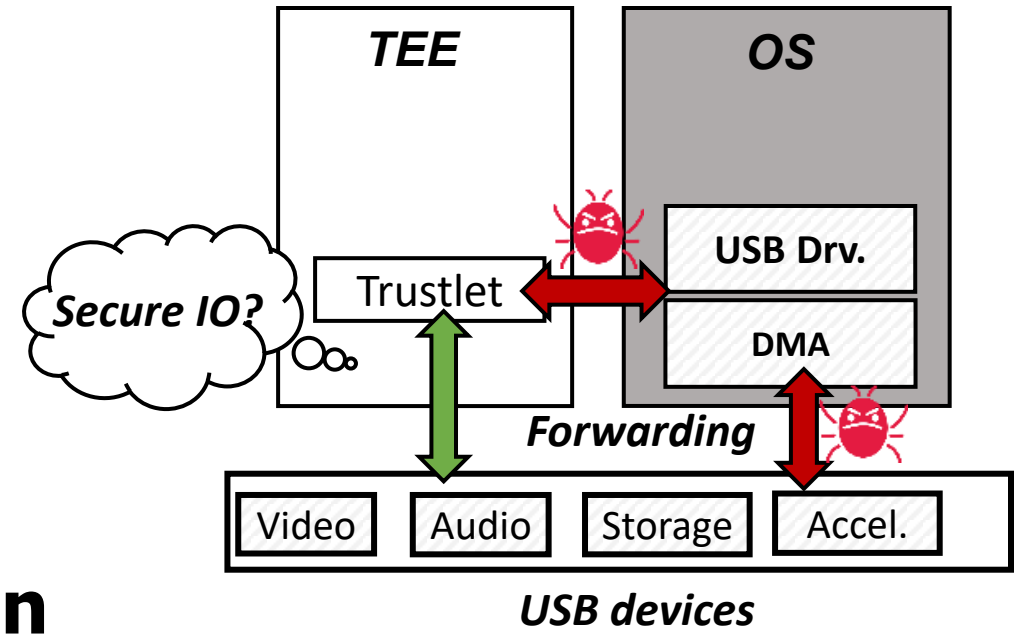
OPTEE-OS issue #1499



Yearly OPTEE OS commits of device drivers

The USB Secure I/O Dilemma (contd.)

- **Important, yet missing USB support**
  - **Most diversely, widely deployed I/O devices**
  - **Absent from commodity TEEs**
- **Can we forward to OS instead?**
  - A common practice
  - … but fails for secure IO
- **Reason: no end-to-end encryption**
  - USB as a common transport *by design*



How to *safely* & *practically* bring USB drivers into a TEE?



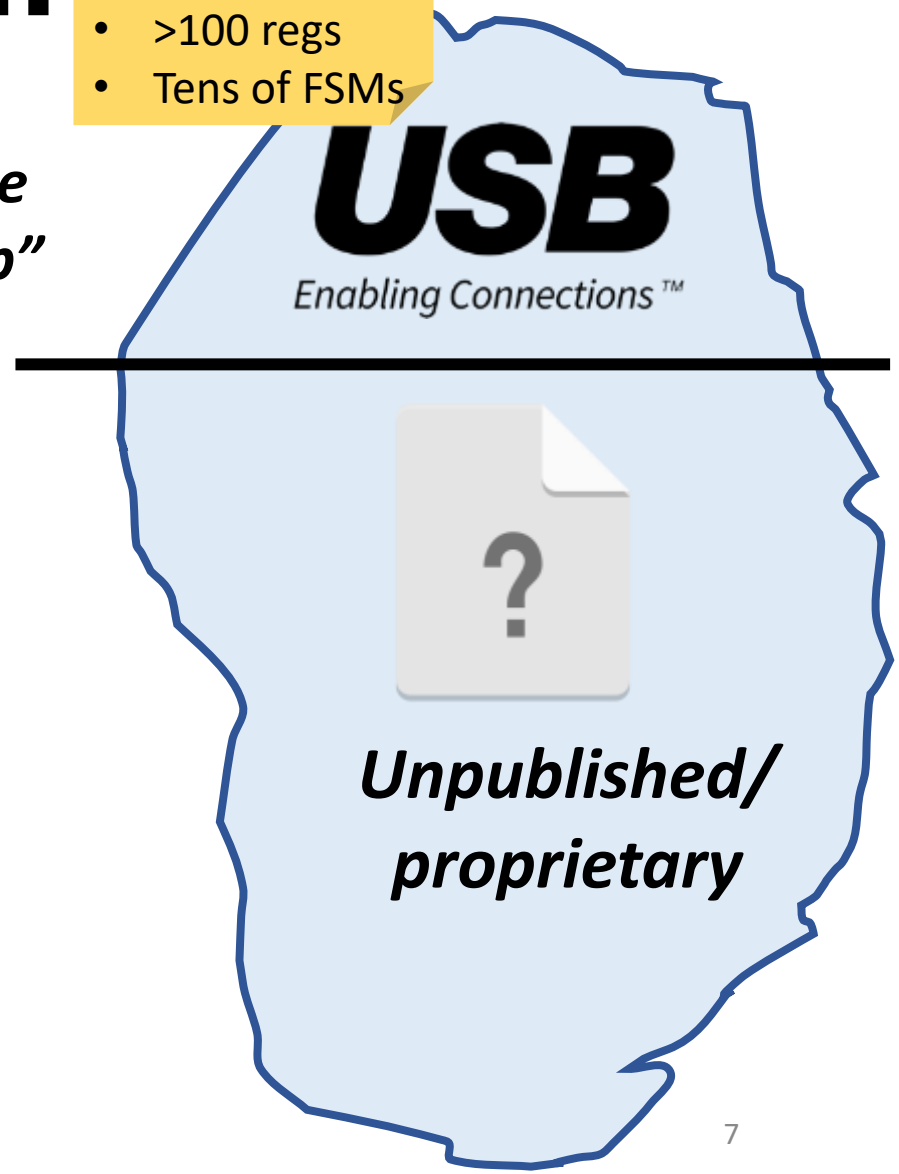
Design space exploration

- **Build from scratch**

- **The tip: large software stack**
  - Protocols, FSMs, Registers
  - Class-specific

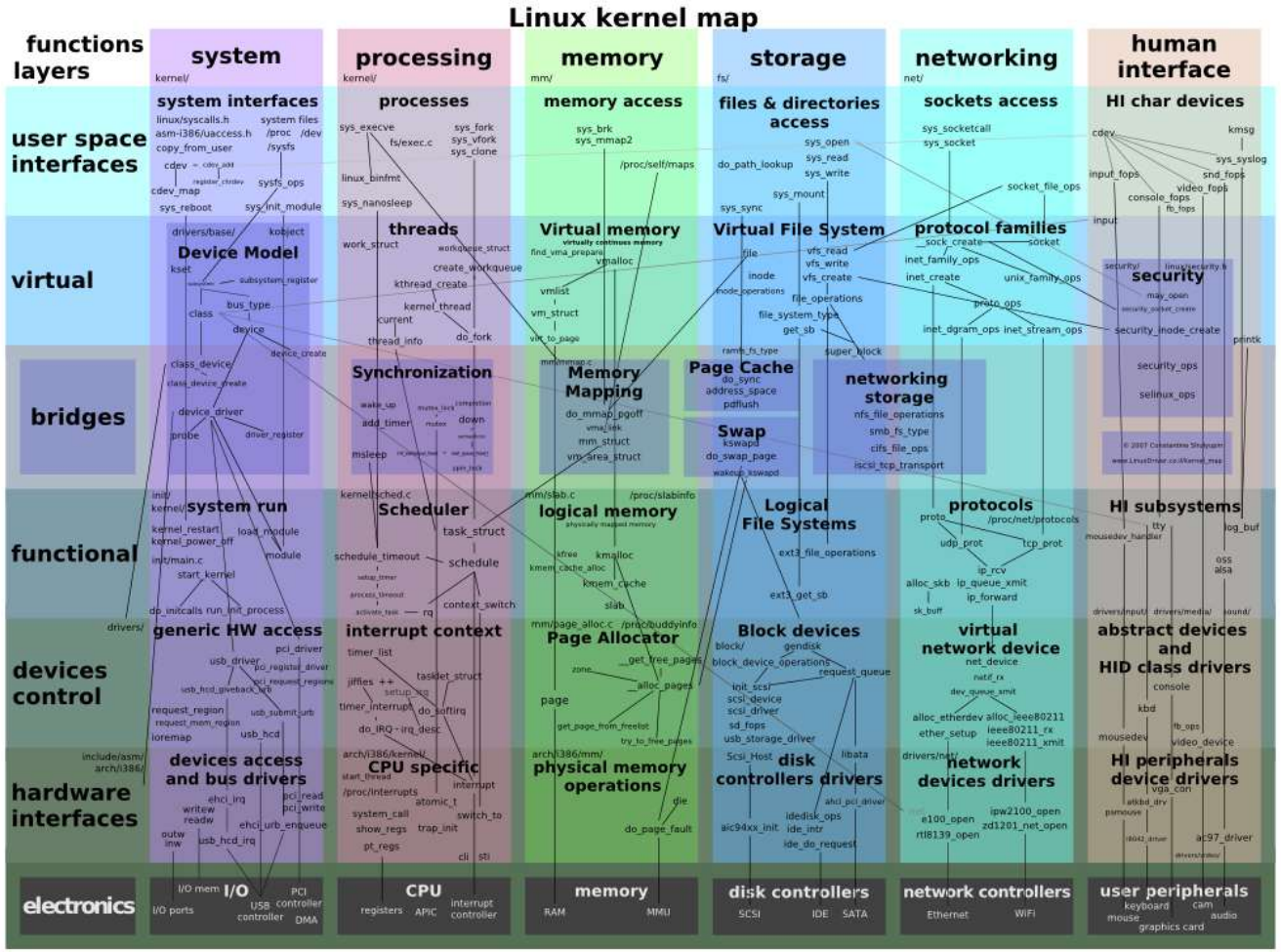
- **Below the surface**
  - Vendor-/model-specifics
  - Unpublished details

- >600 pages of specs
- >100 regs
- Tens of FSMs



Design space exploration

- **Porting**
  - Bundle the full driver with its kernel dependencies
  - End up with whole kernel
- **Trim down**
  - Carve out the only needed functions from kernel
  - Deep device knowledge

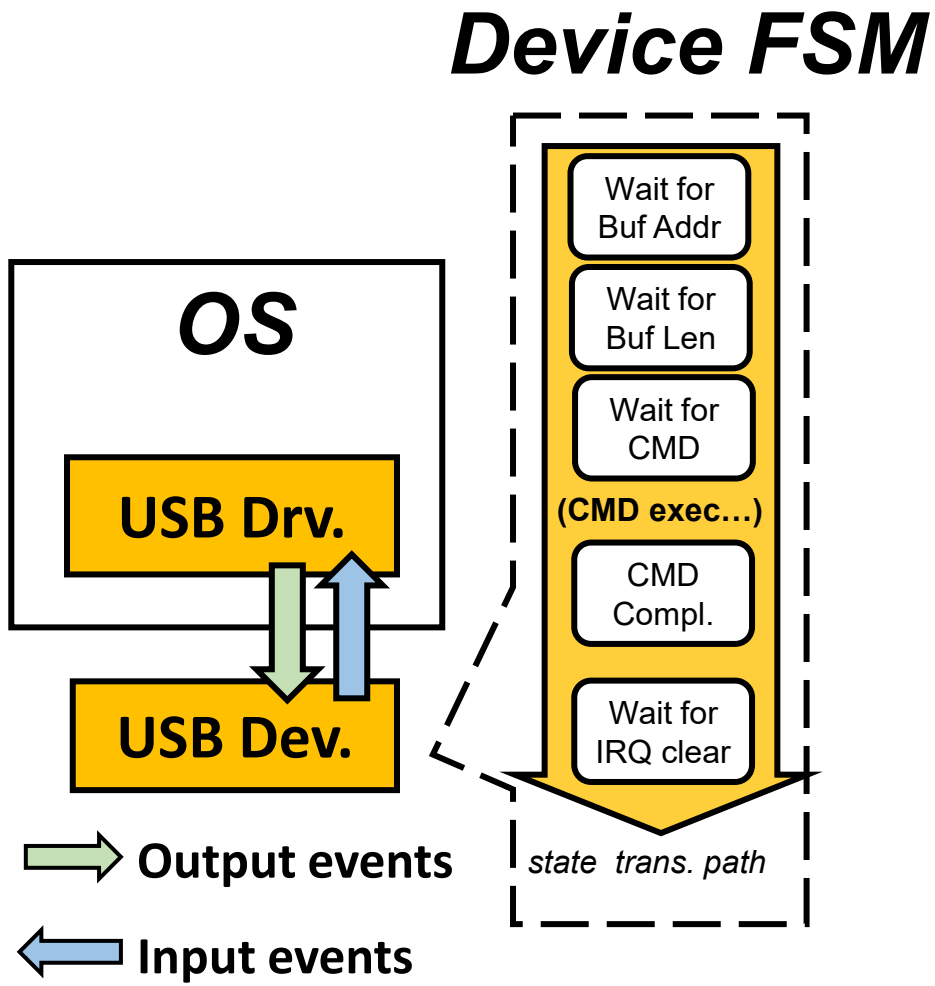


Check out paper for full discussions & details

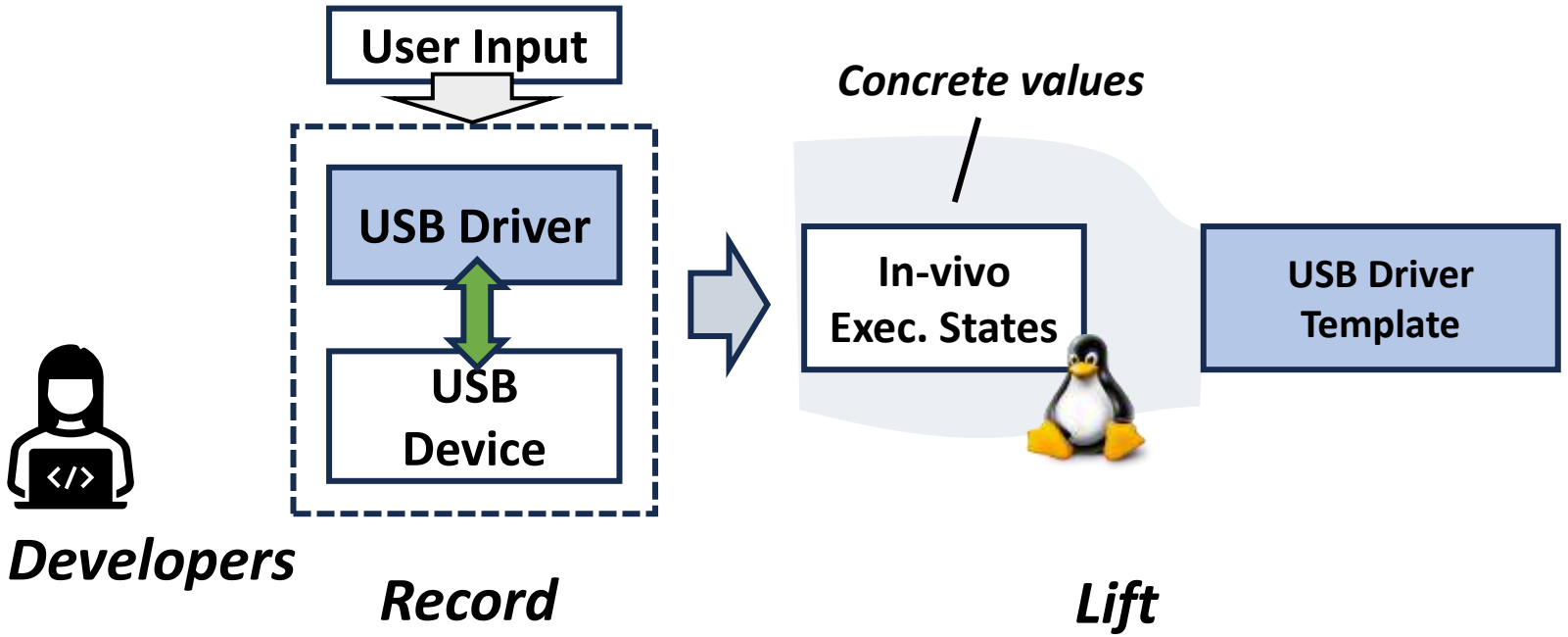
Our approach: Insights

- **Key observations:**
  1. Deterministic, data-independent FSMs
  2. The beaten state transition path
  3. The input/output events

- **Insight**: reuse low-level interactions but not code



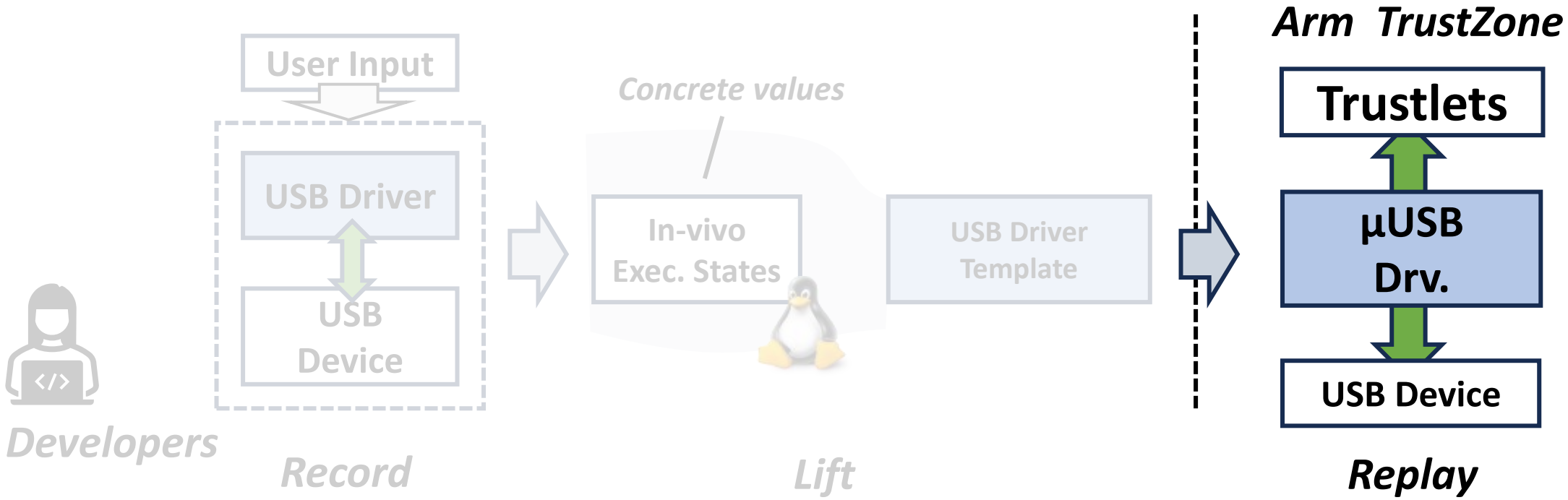
Our approach: µUSB



Approach overview: *Record, Lift, and Replay*

- Record: trace *concrete* USB executions
- Lift: symbolize concrete values for generalizability

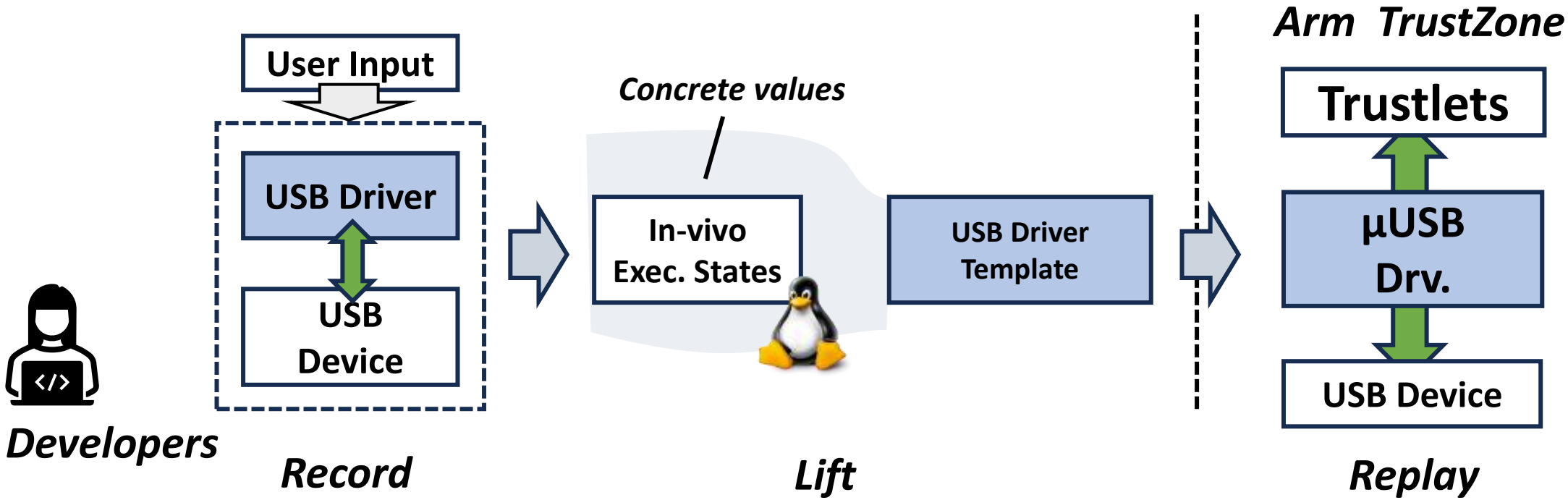
Our approach: µUSB (contd.)



Approach overview: *Record, Lift, and Replay*

- Record: trace *concrete* USB executions
- Lift: symbolize concrete values for generalizability
- Replay: reproduce the same USB IO function

Our approach: Benefits



Approach overview: *Record, Lift, and Replay*

Benefits

Safety (Fully isolated) & Practicality (Ex-vivo processing)

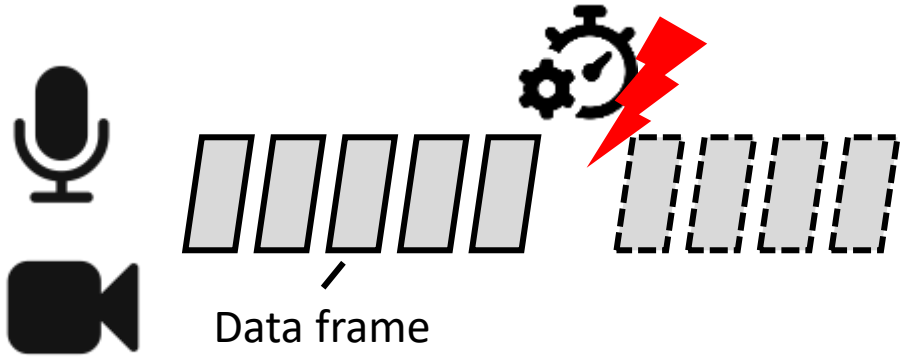
Challenges: How & Why μUSB works?

- **Completeness**
  - What & how to trace necessary and sufficient in-vivo execution states from USB driver execution?

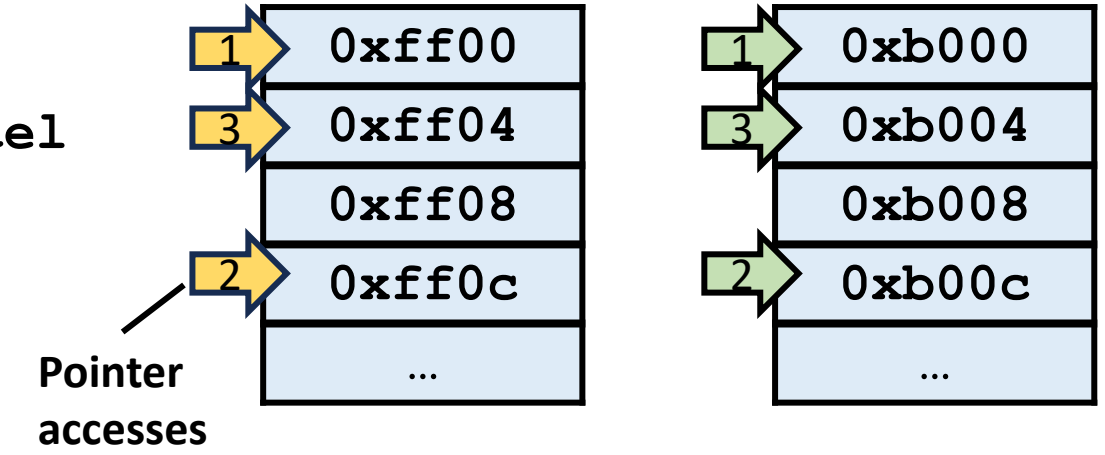
- **Expressiveness**
  - Conctrete tracing → Symbolized template?

Challenge I: completeness

- **Problem:**
  1. Timing-critical USB isochronous transfers
     - For streaming video & audio

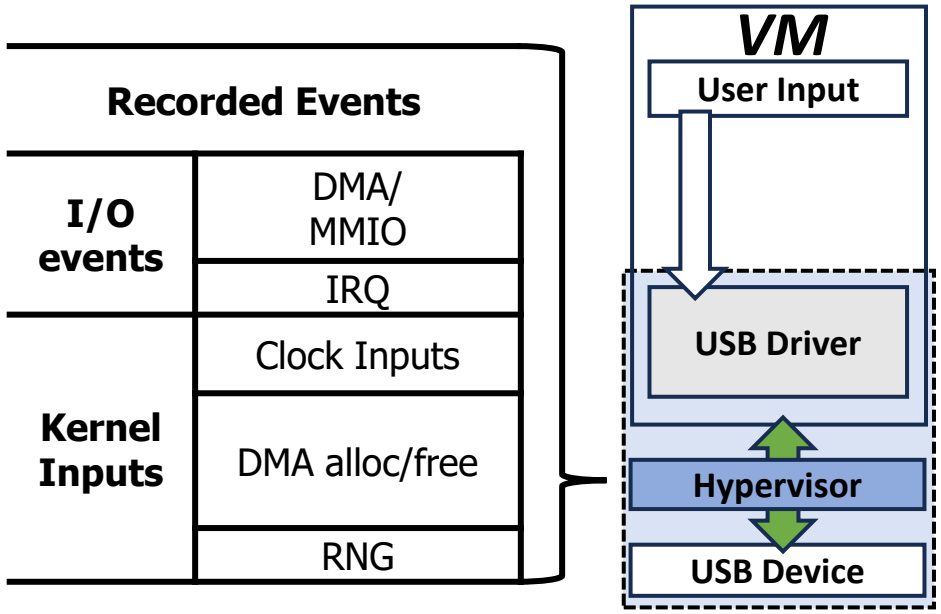


  2. Accesses mainly in DMA pointers
     - No register accessors, e.g., **`readl`**, **`writel`**
     - Runtime variations of addresses



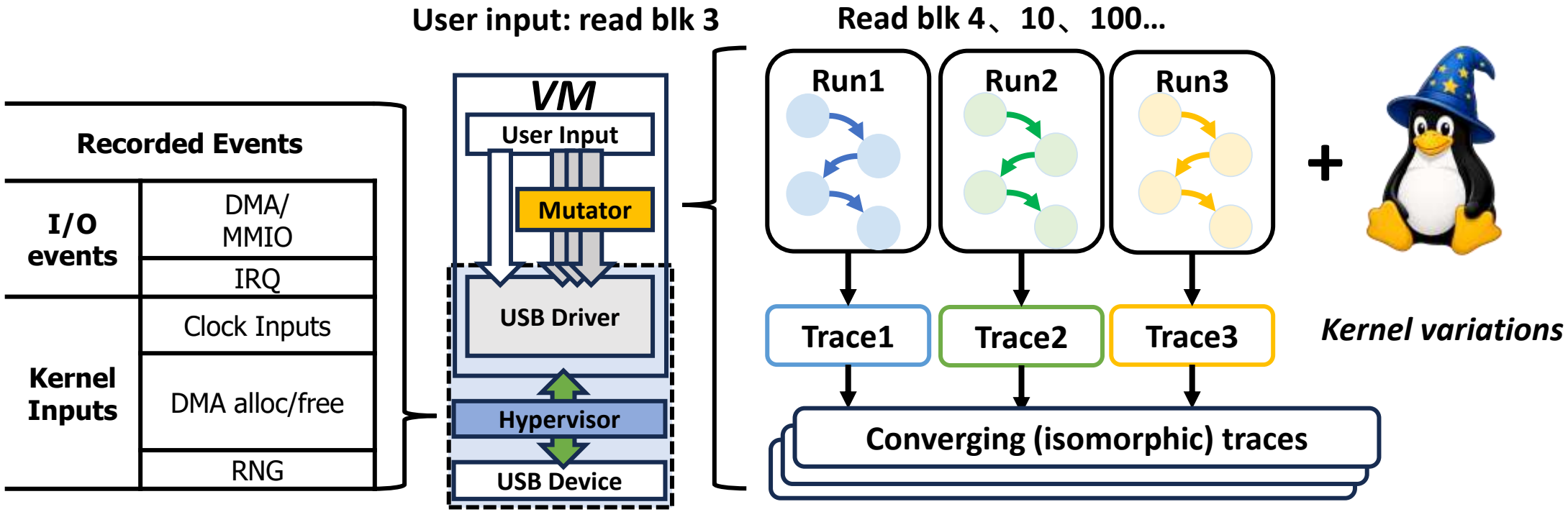
Challenge I: completeness

- **Sol: <u>Mutational Recorder</u>**
  - A lightweight VM to trap all **low-level events** at stage 1 (i.e., hypervisor) plus **kernel trace**



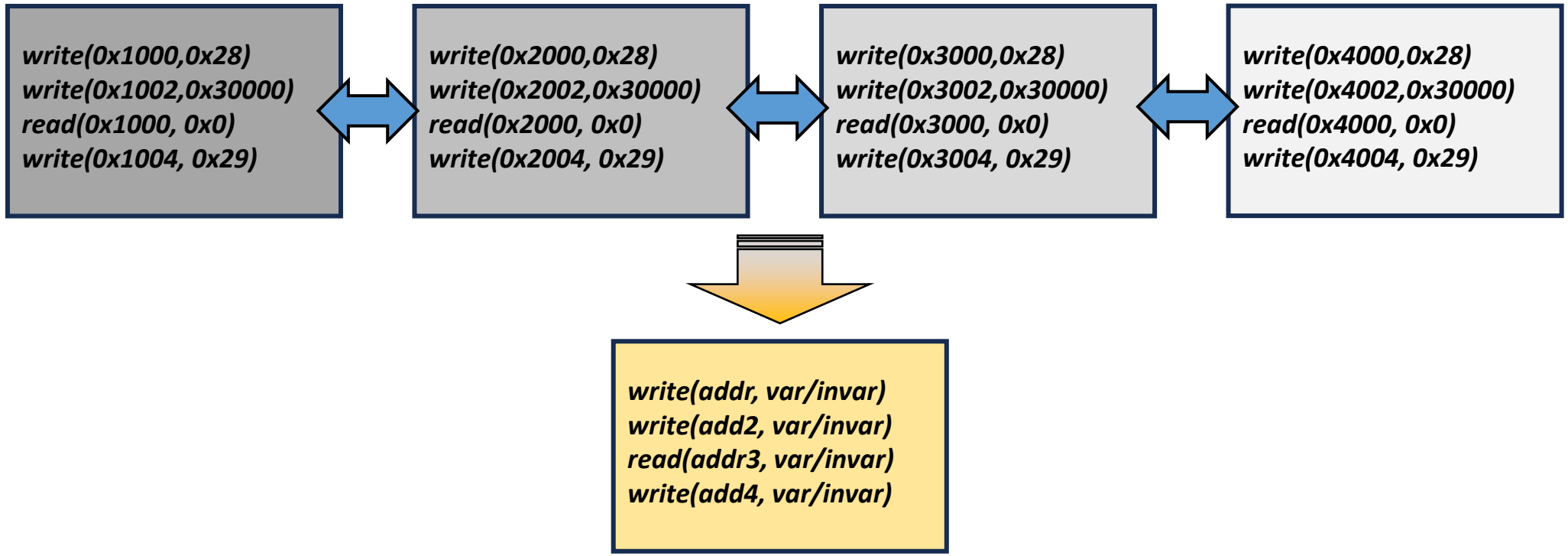
Challenge I: completeness

- **Sol: <u>Mutational Recorder</u>**
  - A lightweight VM to trap all **low-level events** at stage 1 (i.e., hypervisor) plus **kernel trace**
  - **Mutate** user and kernel inputs to expose runtime variation until *convergence*



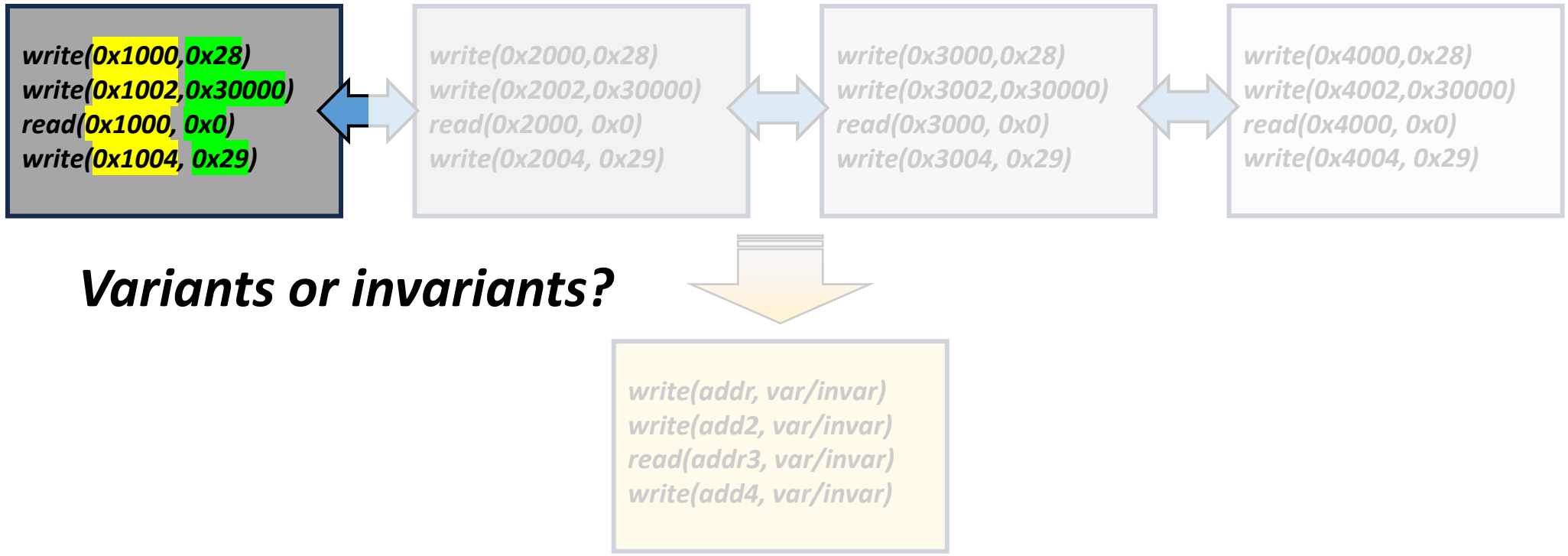
Challenge II: expressiveness

- **Problem: concrete values → programmatic forms**



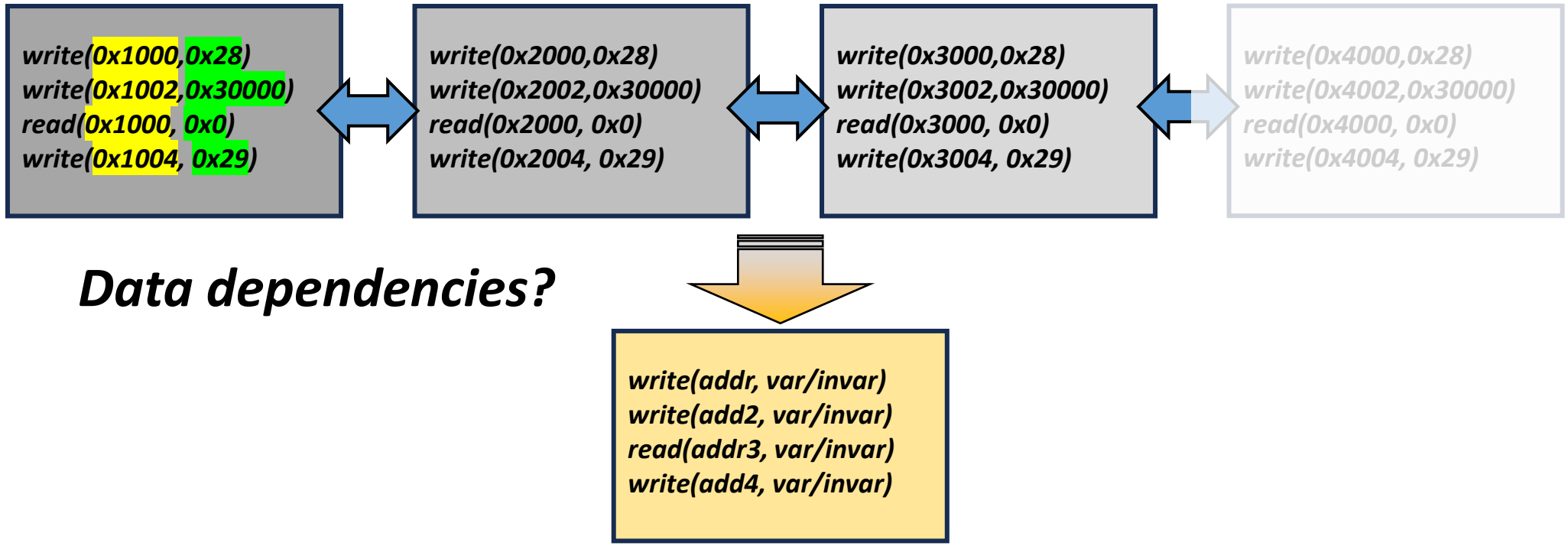
Challenge II: expressiveness

- **Problem: concrete values → programmatic forms**
  - Distinguish variants v.s. invariants from concrete traces



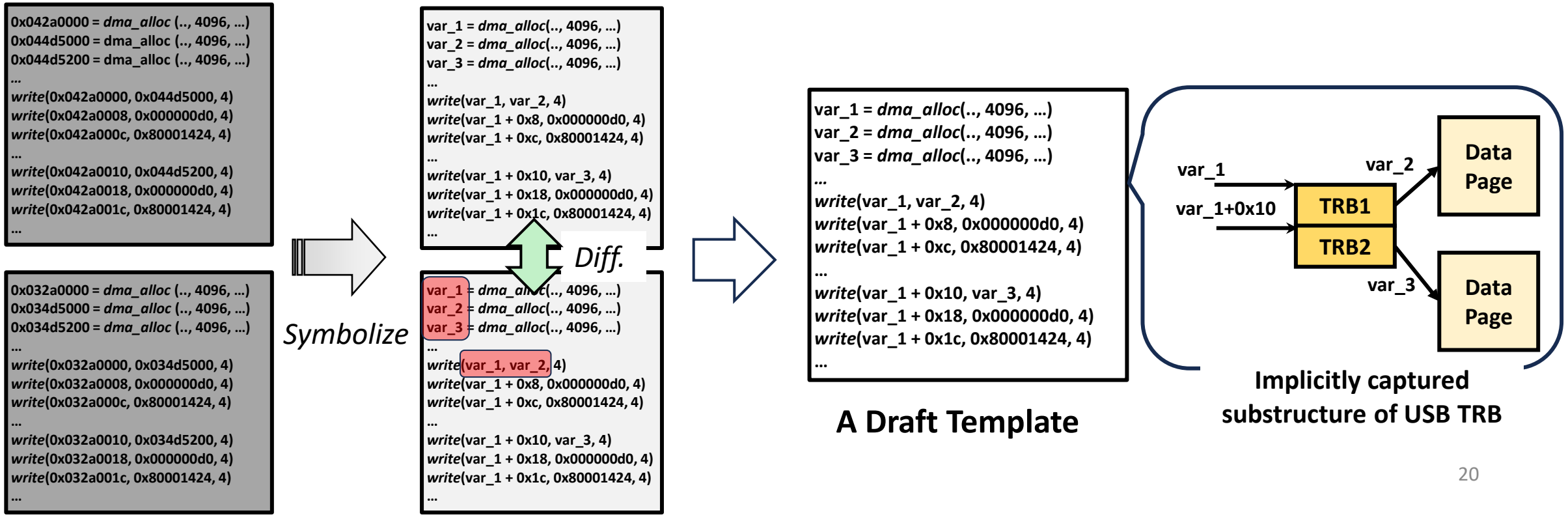
Challenge II: expressiveness

- **Problem: concrete values → programmatic forms**
  - Distinguish variants v.s. invariants from concrete traces
  - Recover implicit data dependencies between variants



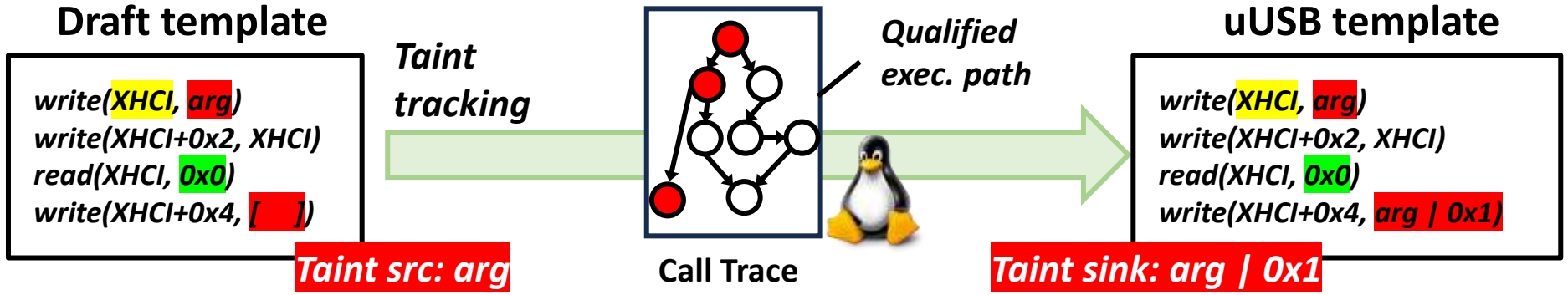
Expressiveness: (in)variants?

- **Sol I: <u>Symbolic differential trace analysis</u>**
  - **Invariants:** unchanged across runs
  - **Variants:** values change but not *substructure!*



Expressiveness: data dependencies

- **Sol II: Qualified Taint Tracking**
  - Taint all variants: User/Kernel/Device inputs
  - Track taints using call traces as the ***path qualifier***
    - The *sharpest* data flow information to make analysis precise and scalable



Implementation & experiences

- Implemented in 4K SLoC based on QEMU and KVM (Linux v6.8) on Rpi5

- **Four major USB classes**
  - USB Mass Storage (**Flash drive**)
  - USB Video (**Webcam x 2**)
  - USB Audio (**microphone** and **speaker**)
  - USB HID (**keyboard** and **mouse**)

- **Post analysis**
  - Fixed-size TRB on SCSI
  - Impl. defect on ISOC buffer allocation
  - TRB of HID with high regularity
  - More fun facts in paper!



Storage	*Video*	*Audio*	*HID*
Flash	480P	Mic	Keyboard
	1080P	Speaker	Mouse

Evaluation

- **Does µUSB…** cut integration effort?
- **Does µUSB…** ensure safety?
- **Do µUSB and its drivers…** have low overhead?

- **Methodology**
  - **Comparisons**
    - **Efforts:** *Build from scratch, Port*
      - Quantified efforts: **#** of Opcodes/Configs/Struct, **pages** of specifications
    - **Performance:** *Native, Circle (HID & Storage), Driverlet (reference point)*
  - **Benchmarks**
    - **USB Mass Storage:** FIO
    - **USB Video/Audio:** V4L2, ALSA
    - **USB HID:** synthetic presses/moves

µUSB saves developer efforts

Build from scratch

- Hundreds of page specs + several months
- Missing documents

	Opcode	Spec.	Conf. Desc.	Data Struct.
xHCI	**17**	**645**	**Nah**	**421**
USB Stor.	**13**	**554**	**3+**	**50**
USB Video	**8**	**169**	**93+**	**31**
USB Audio	**11**	**239**	**18+**	**36**

Port

- 100+ diverse functions from over 10 subsystems
- At least 5K SLoC

µUSB

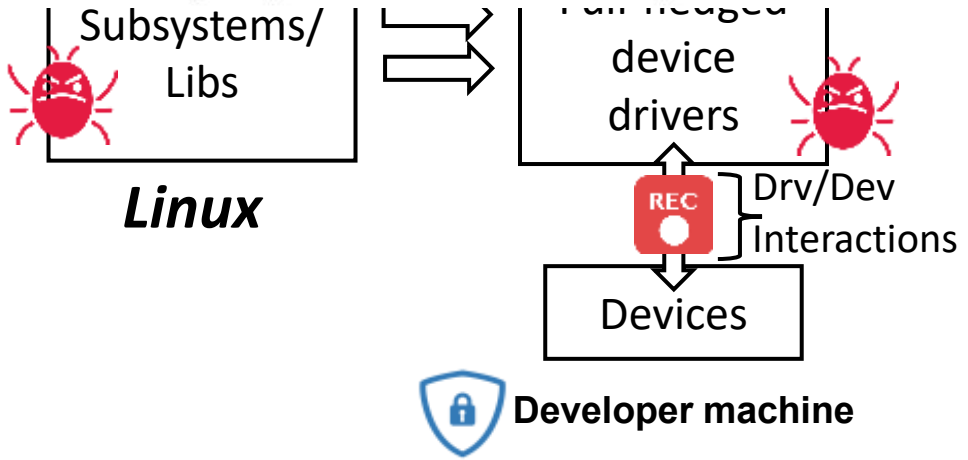
- **One-time efforts (a few weeks)**
- Several minutes

USB Class	IO Function	# Events after Lifting
Stor.	Read(sector, 8, buffer)	37
	Write(sector, 8, buffer)	37
Vid.	Capture(10, 1080P, buffer)	59
	Capture(1, 480P, buffer)	51
Aud.	Capture(2, 10, S16_LE, 44100, buffer)	51
	Playback(2, 10, S16_LE, 44100, buffer)	51
HID	Keystroke(50)	21
	MouseMove(50)	21

Security analysis

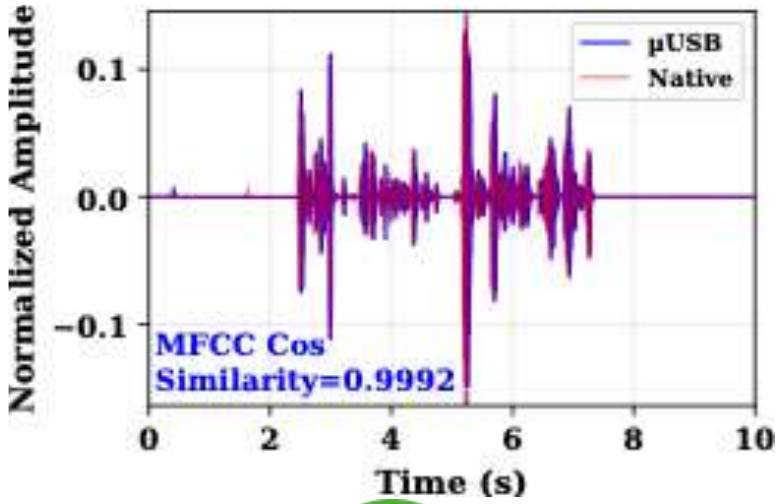
µUSB's Design	Example CVEs	Description	Effect	Vuln.
Eliminate kernel dependencies	CVE-2023-53580 (N/A)	Deadlock during driver unbind procedure [67].	Kernel Panic (DoS)	Kernel.A
	CVE-2024-50302 (Med)	Uninitialized memory in HID core dependency [71].	Kernel Info Leak	Kernel.C
	CVE-2024-57981 (Med)	Null pointer dereference in command abort logic [75].	Kernel Panic (DoS)	Kernel.A
No original USB driver code	CVE-2024-26659 (Med)	Incorrect xHC ownership assumption leads to UAF [68].	Memory overwrite	Kernel.I
	CVE-2024-30212 (High)	Unchecked parameter allows arbitrary RAM r/w [70].	Arbitrary mem r/w	Driver.I/C
	CVE-2024-53197 (High)	OOB access in ALSA legacy quirks [73].	Arbitrary mem r/w	Kernel.I
	CVE-2024-57980 (High)	Double-free in uvcvideo status URB handling [74].	Arbitrary code execution	Kernel.I
	CVE-2025-38680 (N/A)	OOB read due to missing len check [76].	Unauthorized mem access	Kernel.I
Isolate normal OS	CVE-2023-34403 (N/A)	Network service confuses OS to access USB files [66].	Info Leak	OS.I
	CVE-2024-27059 (Med)	Malicious device triggers divide-by-zero [69].	Kernel Panic (DoS)	OS.A
	CVE-2024-53150 (High)	Out-of-bounds read in ALSA USB sound driver [72].	Info Leak	OS.I

I: Integrity; C: Confidentiality; A: Availability



Correctness analysis

- ✓ **Statically vetting**
  - Verify call traces
  - Check w/ specs.
- ✓ **IO data integrity validation**



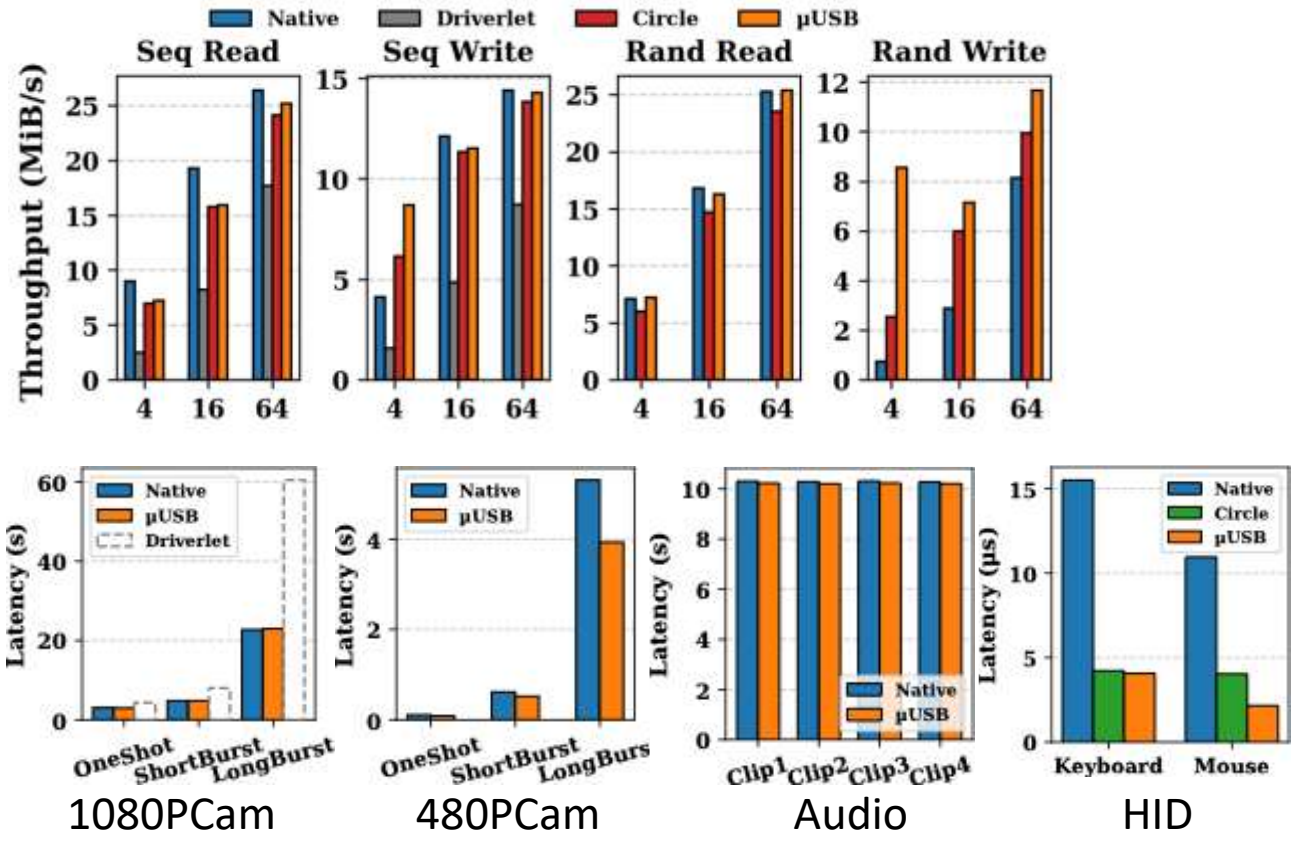
- ✓ **Stress testing**
  - Over 2 weeks
  - 24 hours capture for Video/Audio

We retrofit Labelled Transition System (LTS) to give a more formal reasoning in the paper!

µUSB incurs low overhead

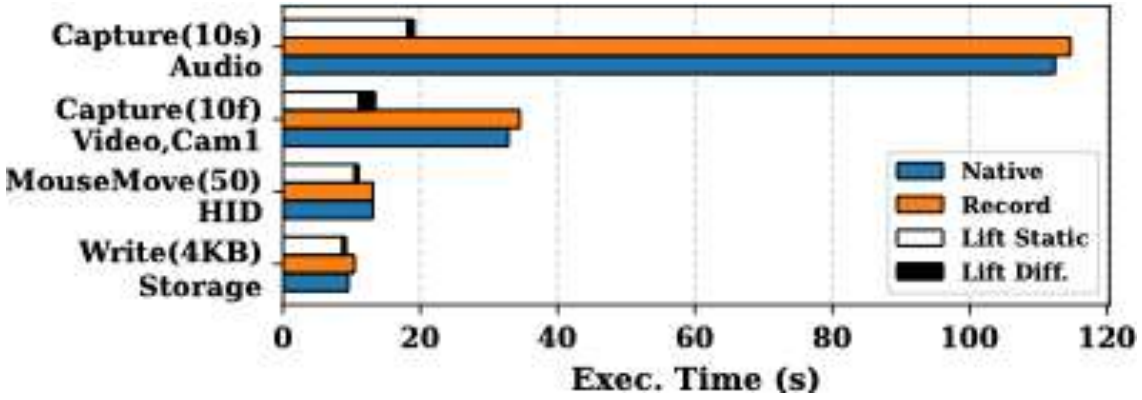
- **On four classes**
  - **Storage:** 16.26/11.01 MiB/s
    - Native (17.42/6.68 MiB/s)
  - **Camera:** 2964 and 87 ms/frame
    - 3% and 20% faster than native
  - **Audio:** close to native for its rate
  - **HID:**
    - 4.5× faster than ***Native***
    - 1.3× faster than ***Circle***

- **Throughput close to native; latency much better**



µUSB incurs low overhead (contd.)

- **Fast USB driver generation**
  - Avg. 56.9 seconds
  - Up to 135.1 seconds
  - Most time spent on recording



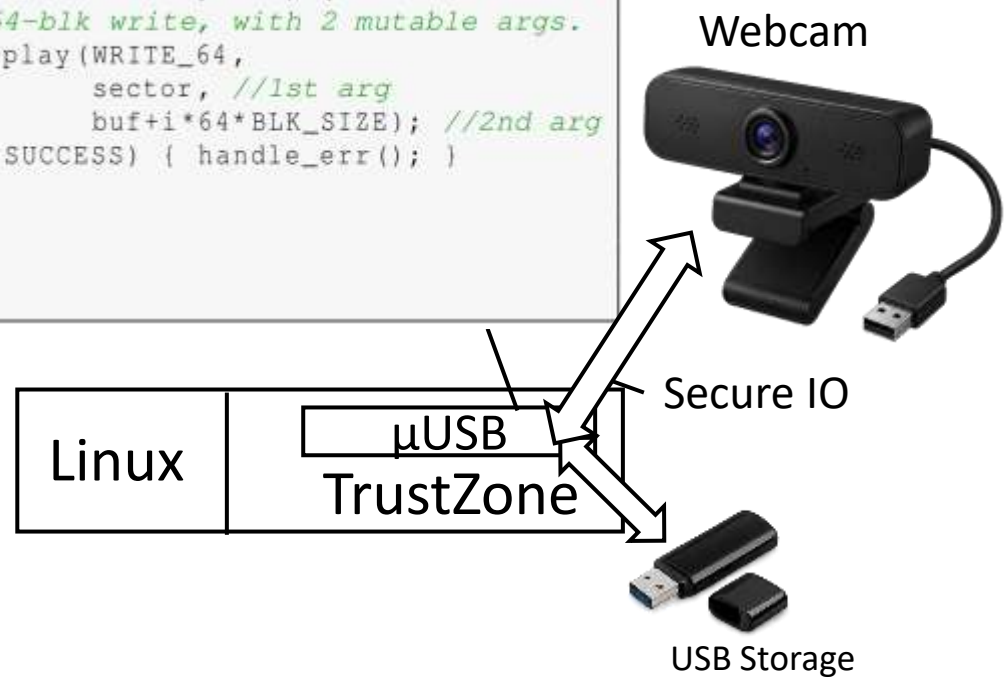
- **Memory friendly**
  - Two orders of magnitude smaller than native drivers.

Dev. Class	Ours	Native	Reduction
Storage	**24KB**	**2.5MB**	**107×**
Video	**400KB**	**4.6MB**	**12×**
Audio	**160KB**	**7.1MB**	**45×**
HID	**15KB**	**1.7MB**	**116×**

µUSB use case

- 1 header file
  - **#include <µUSB.h>**
- 2 templates
  - **VIDEO_CAPTURE**
  - **WRITE_64**

```
#include <uusb.h> // uUSB API
void trusted_surveillance_app() {
  // 1. Allocate DMA buffer in TEE
  void *buf = dma_malloc(FRAME_SIZE);
  int sector = 0;
  while (true) {
    // 2. Replay frame capturing; buf is mutable arg.
    int ret = uusb_replay(VID_CAPTURE, buf);
    if (ret != SUCCESS) { handle_err(); }
    // 3. Dump to USB Storage
    int blk = FRAME_SIZE/SECTOR_SIZE;
    int rounds = blk/64;
    for (int i = 0; i < rounds; i++) {
      // recorded 64-blk write, with 2 mutable args.
      ret = uusb_replay(WRITE_64,
                        sector, //1st arg
                        buf+i*64*BLK_SIZE); //2nd arg
        if (ret != SUCCESS) { handle_err(); }
    }
    sector += 64;
  }
}
```



Closing remarks

- **A novel model for USB Support in TrustZone**
  - Record concrete executions, lift them into replayable templates
- **Key techniques**
  - Mutational recording + Symbolic differential analysis + Qualified taint tracking
- **Practical & safe**
  - Scale to diverse USB devices with strong TEE use cases
  - Near-native performance & reduced attack surface

Thanks for listening