1st Workshop on eBPF and Kernel Extensions

RingGuard: Guard io_uring with eBPF

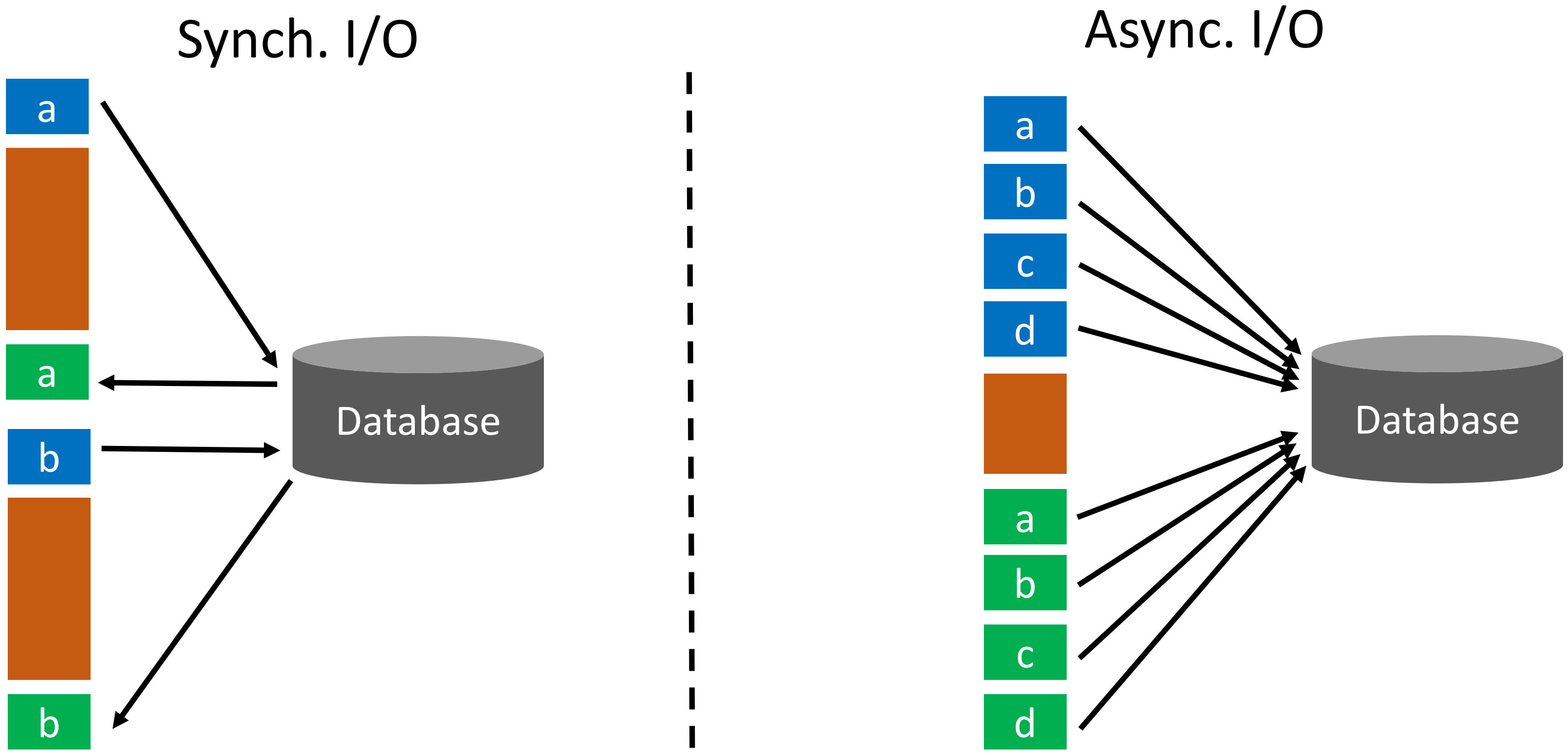
Wanning He*, Hongyi Lu*, Fengwei Zhang, Shuai Wang

SUSTech Southern University of Science and Technology

COMPASS Lab
COMPuter And System Security Lab

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Speed up with asynchronous I/O



🟦 Send request 🟩 Receive respond 🟧 Wait

Asynchronous I/O in Linux

❑ aio

- Only support un-buffered disk I/O
- Blocked if the storage device is not ready
- 104 extra bytes of memory copy are required for each IO event

Asynchronous I/O in Linux

❑ aio

- Only support un-buffered disk I/O
- Blocked if the storage device is not ready
- 104 extra bytes of memory copy are required for each IO event

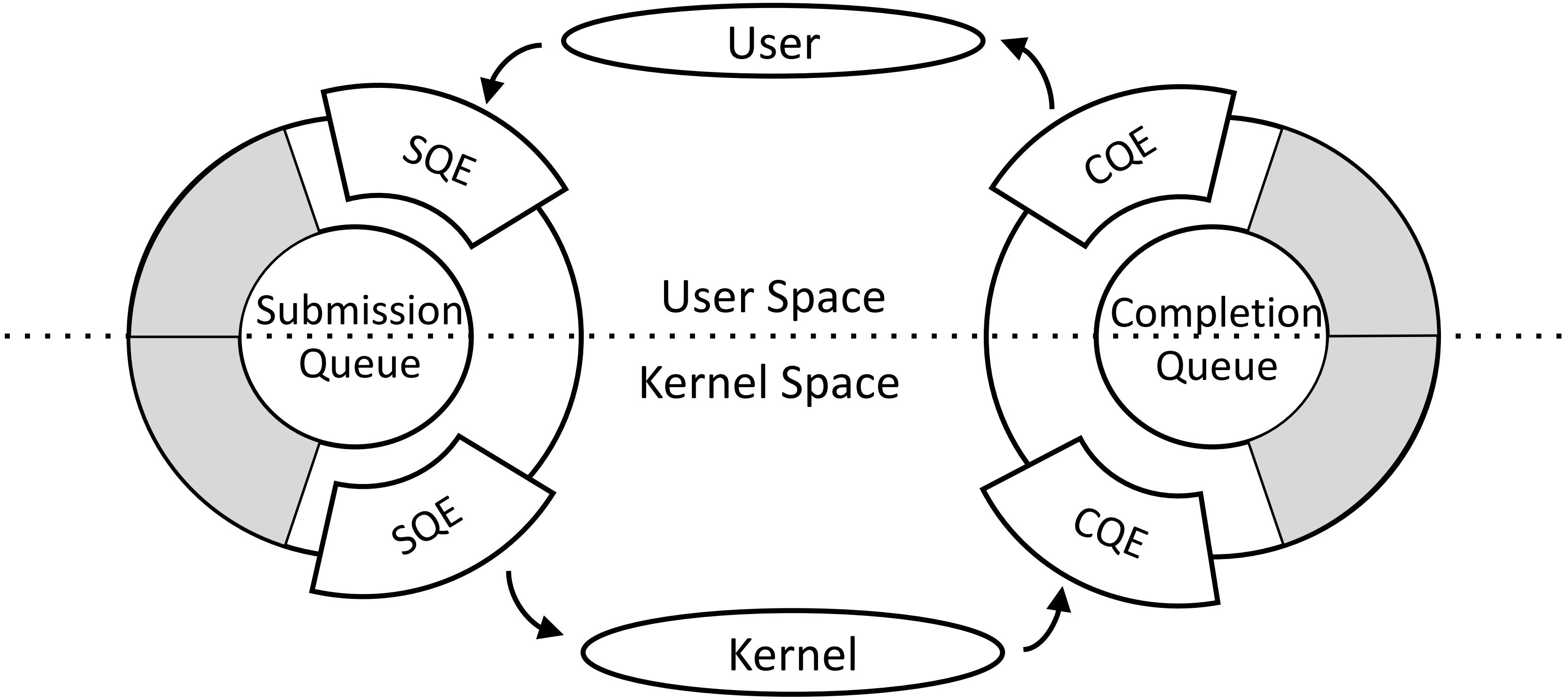
❑ io_uring

- support a wide range of operations
- No need to wait for the file descriptor getting ready
- Could be a zero-copy system; syscall batching

Overview of io_uring

❑ Main components

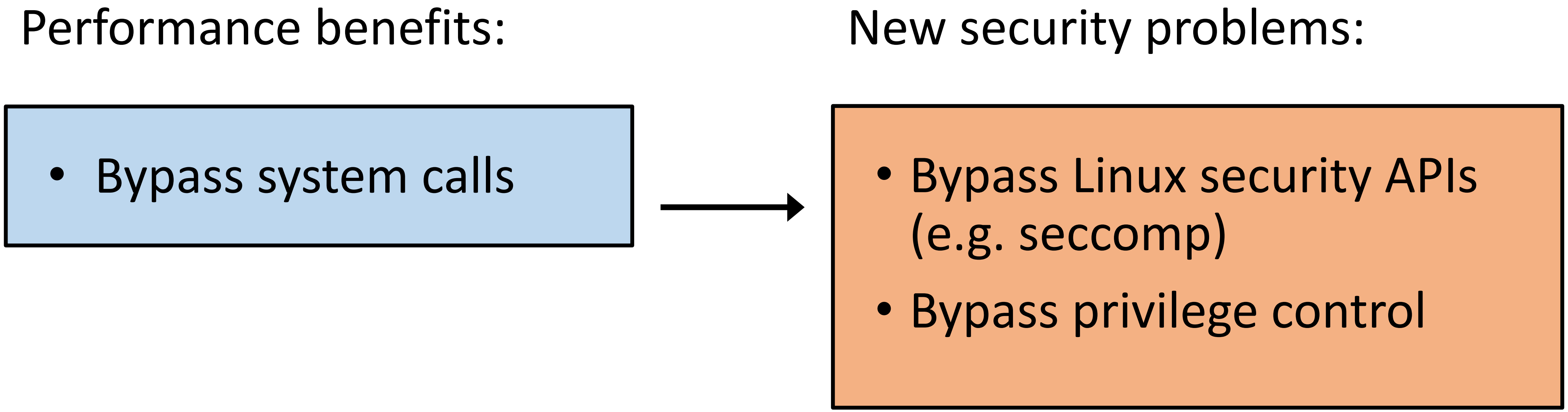
- A submission queue and a completion queue
- User requests are represented as submission queue entries (SQE)
- Their results are represented as completion queue entries (CQE)



Asynchronous I/O with io_uring

- ❑ **Supports a wide range of operations**
  - Disk I/O, network I/O, …
- ❑ **Easy-to-use user-level interface**
  - Can be programmed with C and Rust
- ❑ **Efficiency**
  - Shared memory
  - Syscall batching

io_uring security concerns



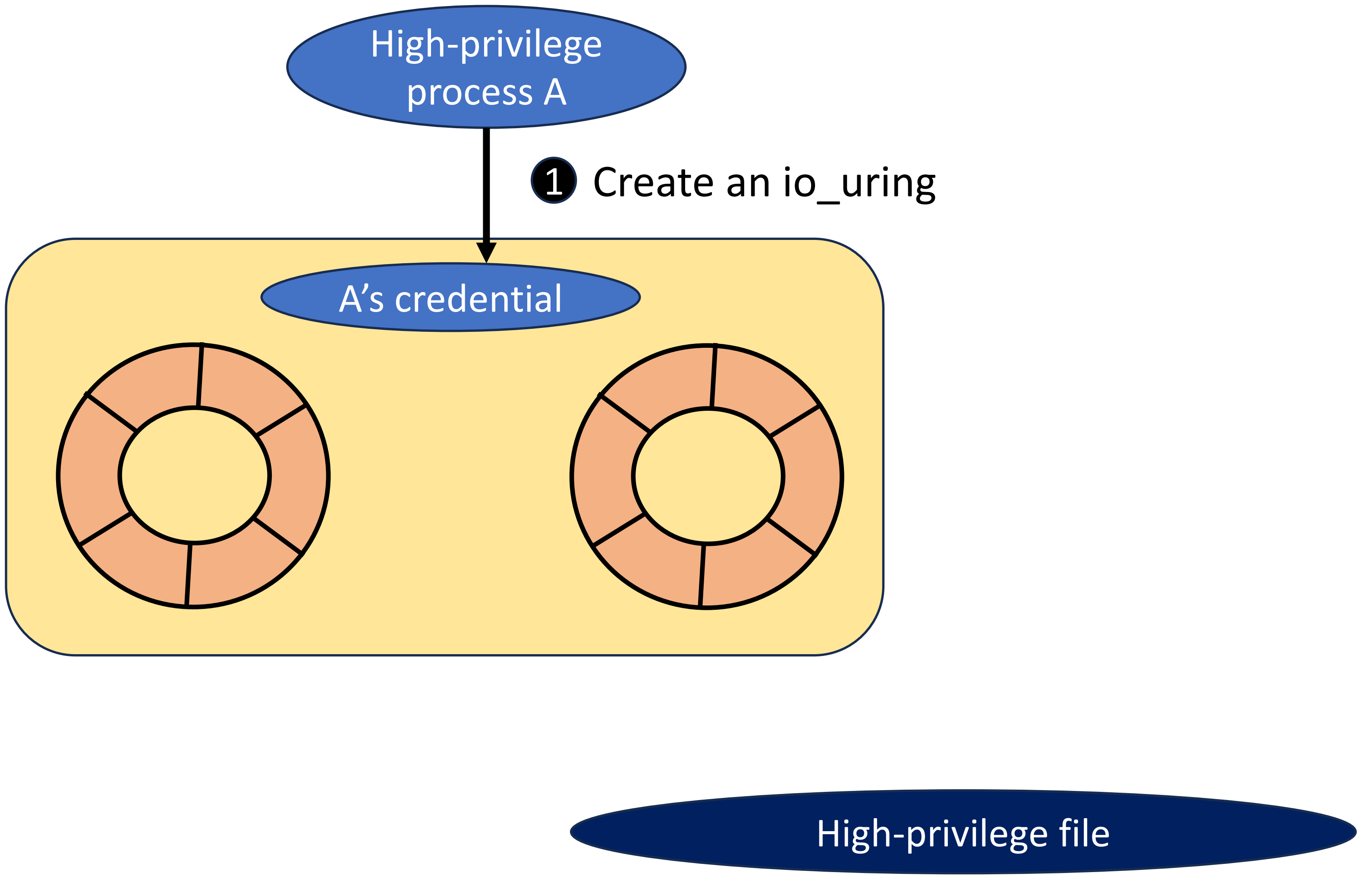
Performance benefits:

- Bypass system calls

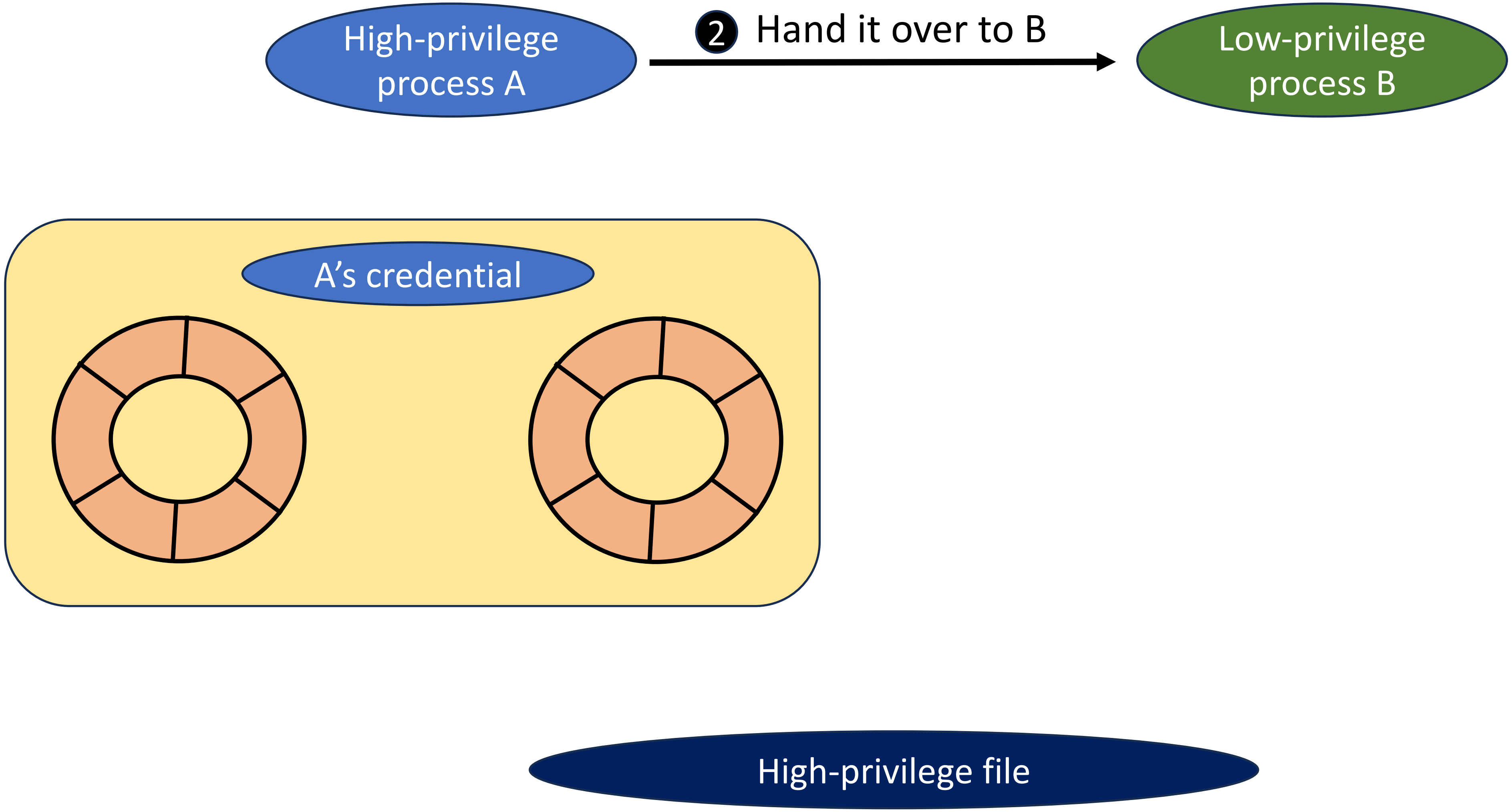
New security problems:

- Bypass Linux security APIs (e.g. seccomp)
- Bypass privilege control

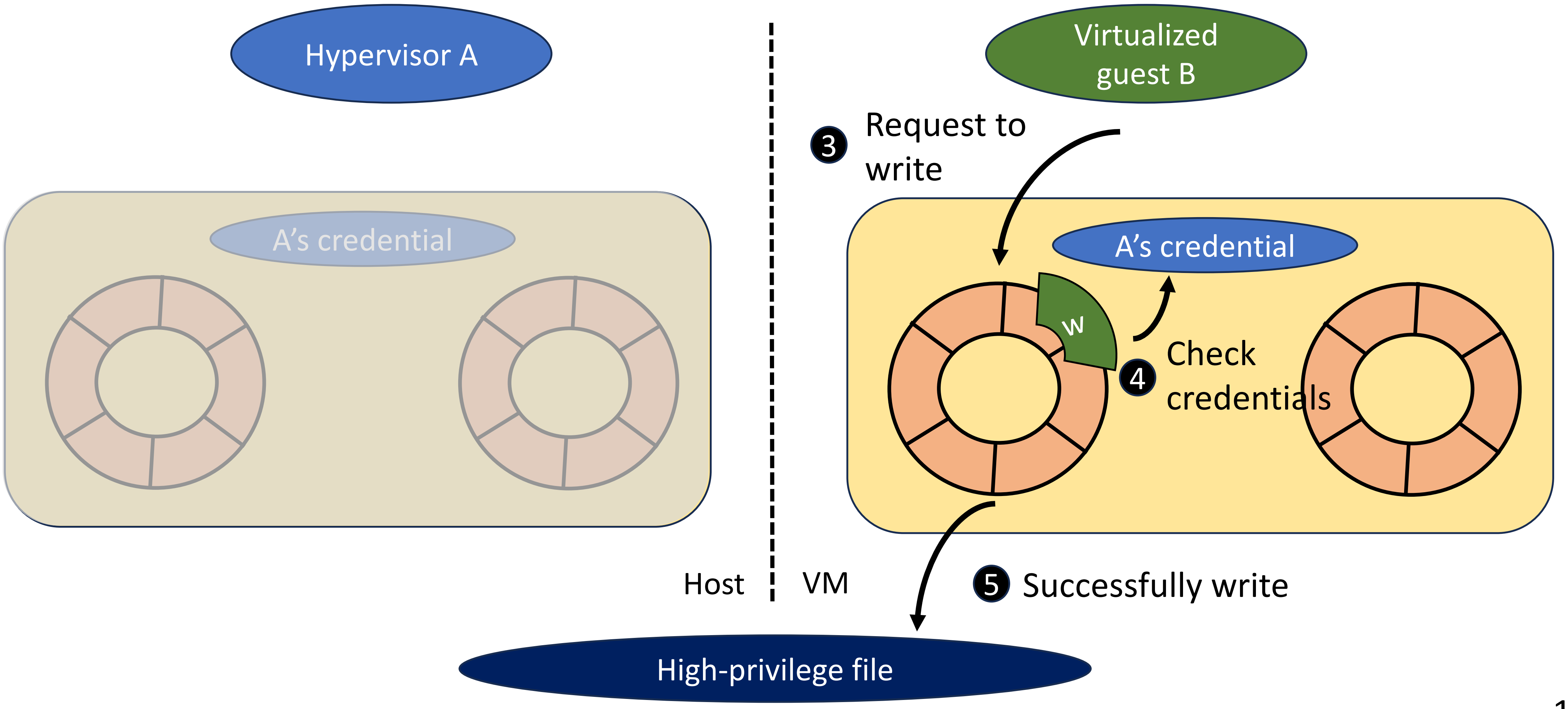
Bypass privilege control using io_uring



Bypass privilege control using io_uring



Bypass privilege control using io_uring



io_uring security concerns

- Increasing number of vulnerabilies are reported

Reported year	#CVEs
2019	1
2020	1
2021	3
2022	10
2023 (Sep.)	11

io_uring security concerns

- Increasing number of vulnerabilies are reported

Reported year	#CVEs
2019	1
2020	1
2021	3
2022	10
2023 (Sep.)	11

eBPF programs can be hooked to an io_uring and verify its operations

Advantages of using eBPF for io_uring protection

- Lightweight, flexible, and transparent
- On-the-fly protection without recompiling/rebooting the kernel
- Independent of hardware features - can be deployed to various scenarios.

RingGuard: our solution to io_uring security issues

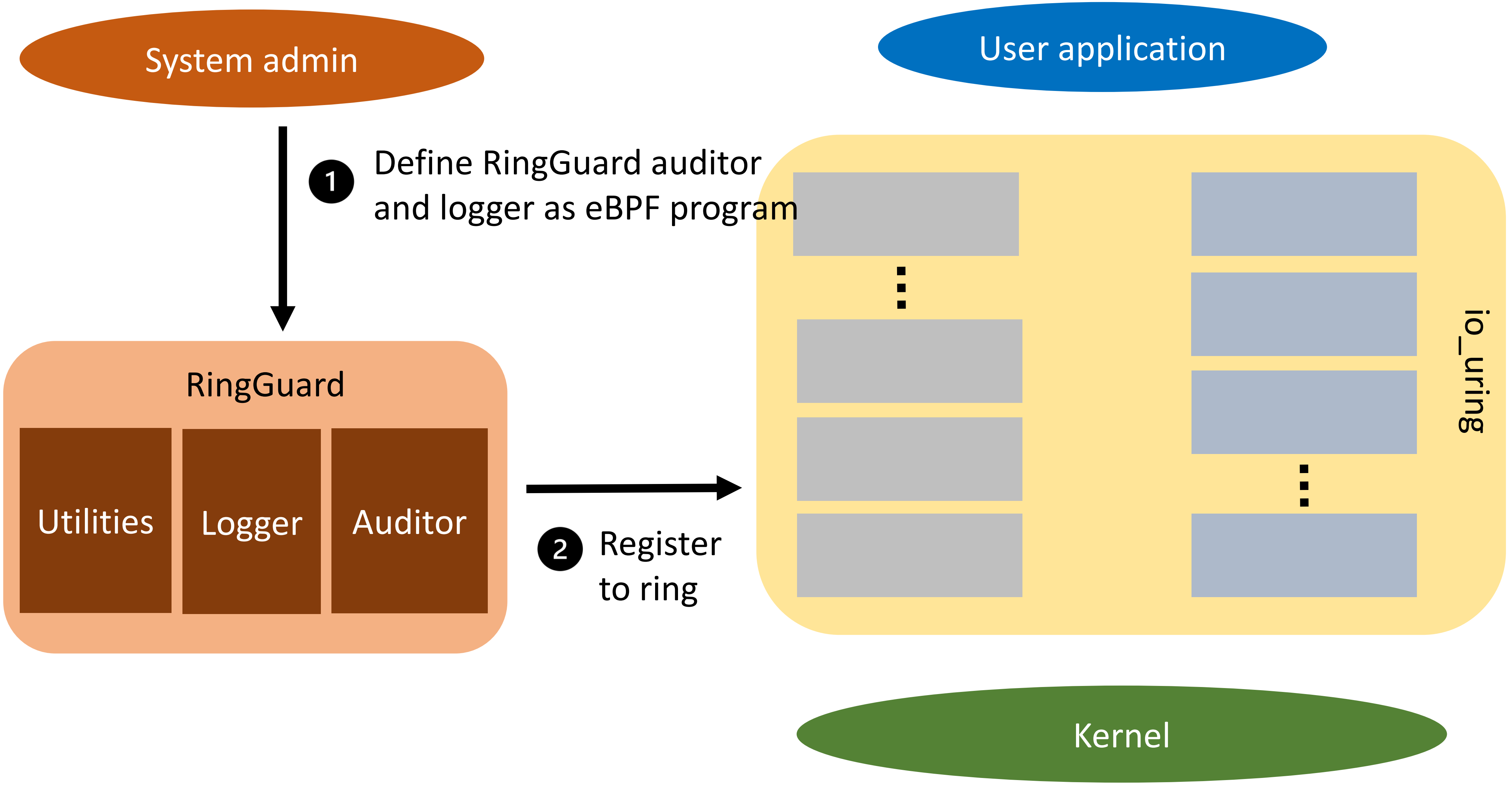
❑ **Key idea:**

- A framework that allows system administrators to define eBPF programs to verify io_uring requests

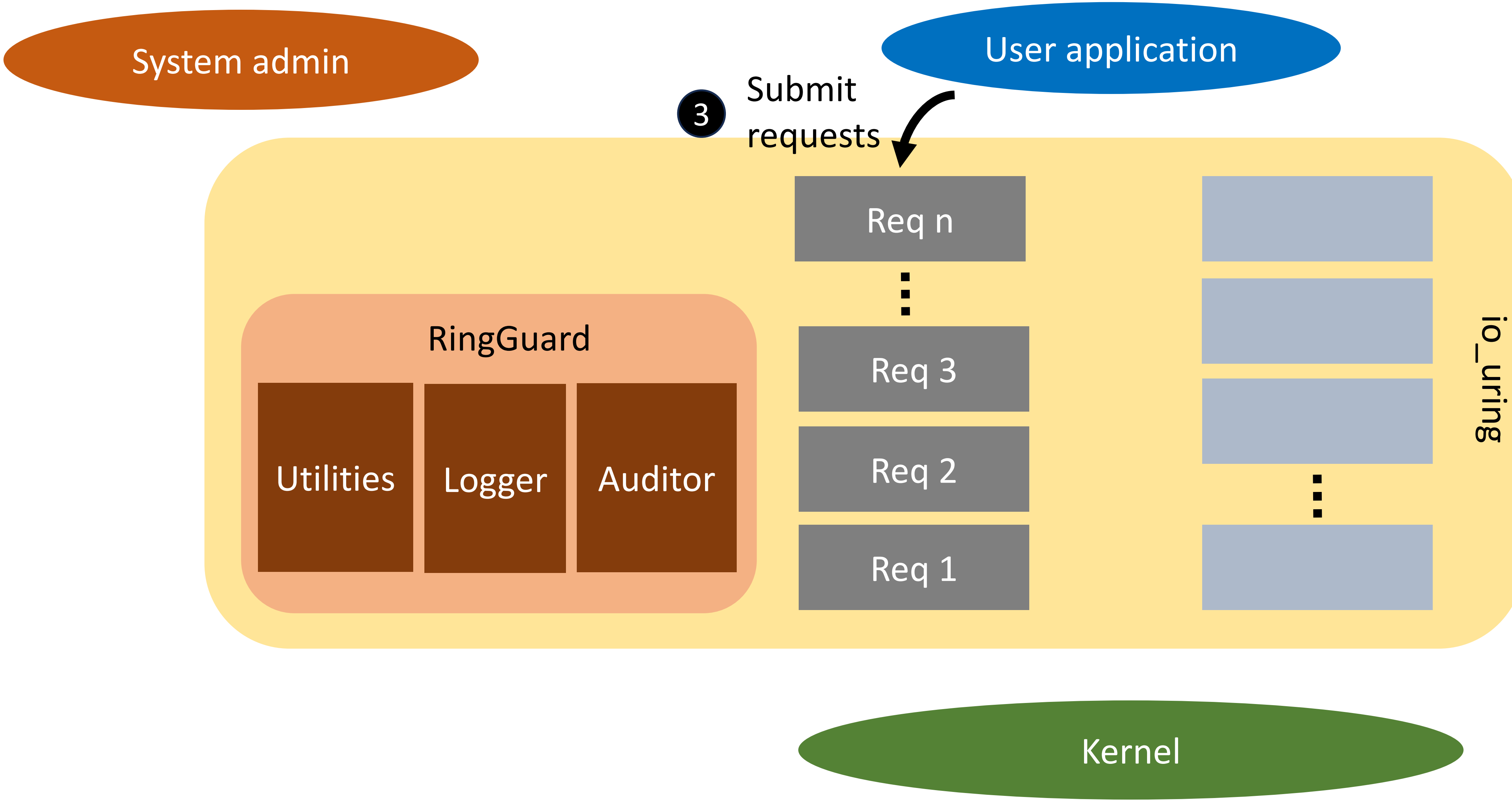
❑ **New extensions to the kernel:**

- Introduce new a BPF hookpoint to the io_uring subsystem
- Necessary helpers for RingGuard eBPF programs

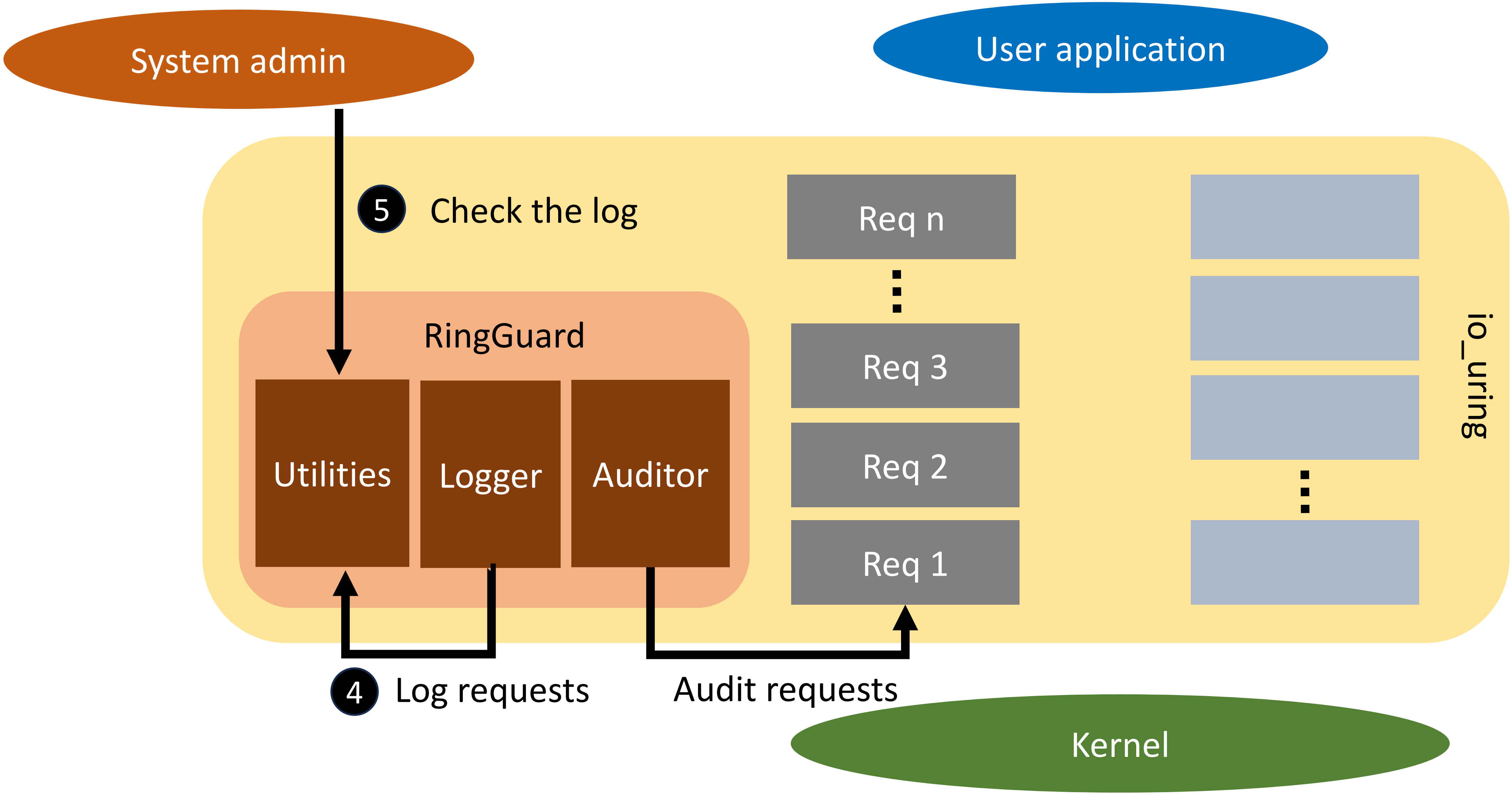
The workflow of RingGuard



The workflow of RingGuard



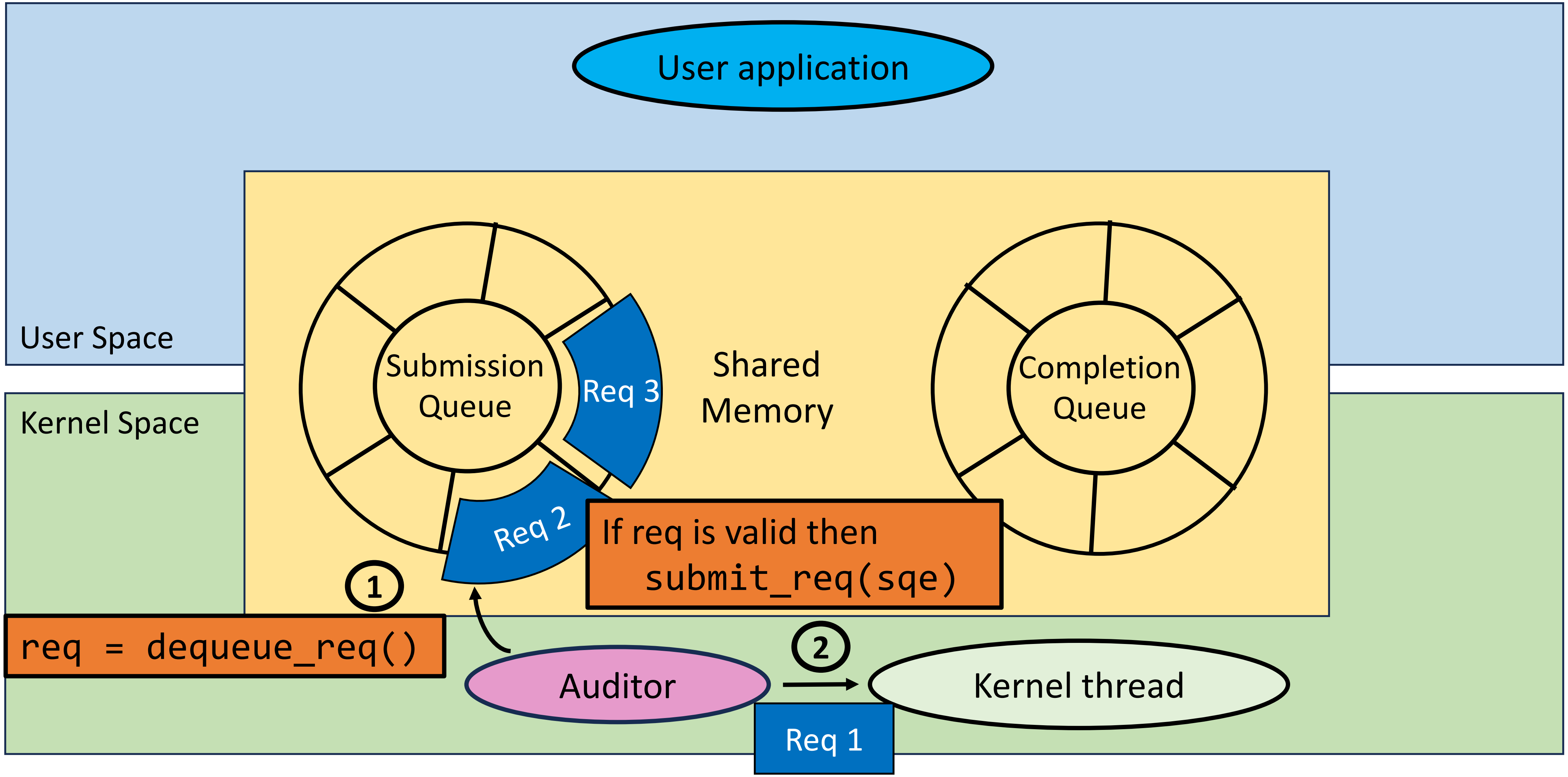
The workflow of RingGuard



Challenge of RingGuard performance

- ❑ Submitting 512 requests simultaneously is 7x faster than separately
- ❑ **Cause:** repeatedly construct & destruct eBPF runtime contexts
- ❑ **Solution:** batch io_uring requests and audit them all at once
  - `threshold`: the minimum number of requests to trigger RingGuard
  - `timeout`: the maximum waiting time if there are not enough requests
- ❑ **Results:** improve the performance by around 17%

Audit io_uring requests with eBPF

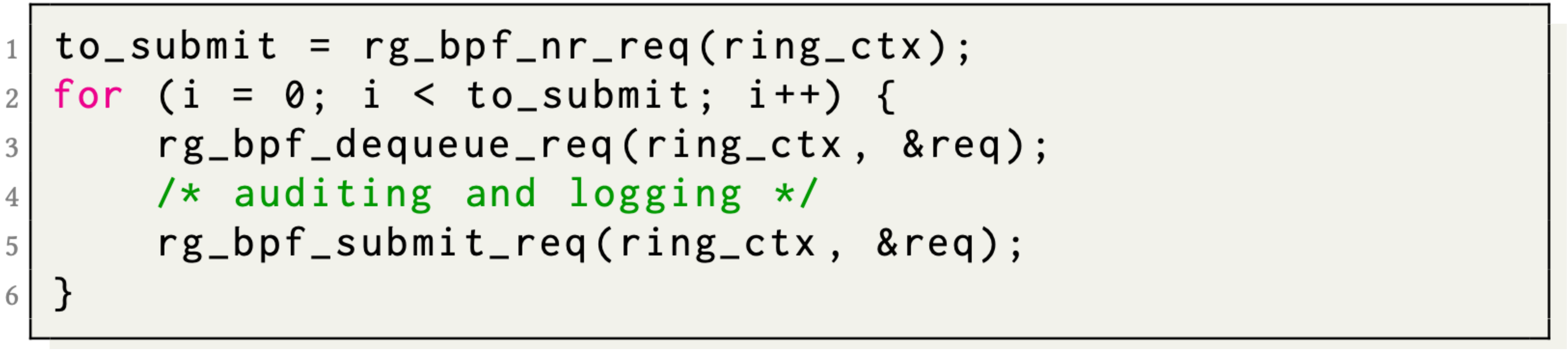


```
req = dequeue_req()
```

If req is valid then
```
  submit_req(sqe)
```

Audit io_uring requests with eBPF

- A RingGuard eBPF program (simplified)



```
to_submit = rg_bpf_nr_req(ring_ctx);
for (i = 0; i < to_submit; i++) {
    rg_bpf_dequeue_req(ring_ctx, &req);
    /* auditing and logging */
    rg_bpf_submit_req(ring_ctx, &req);
}
```

The auditing rule can be flexibly defined by the administrator!

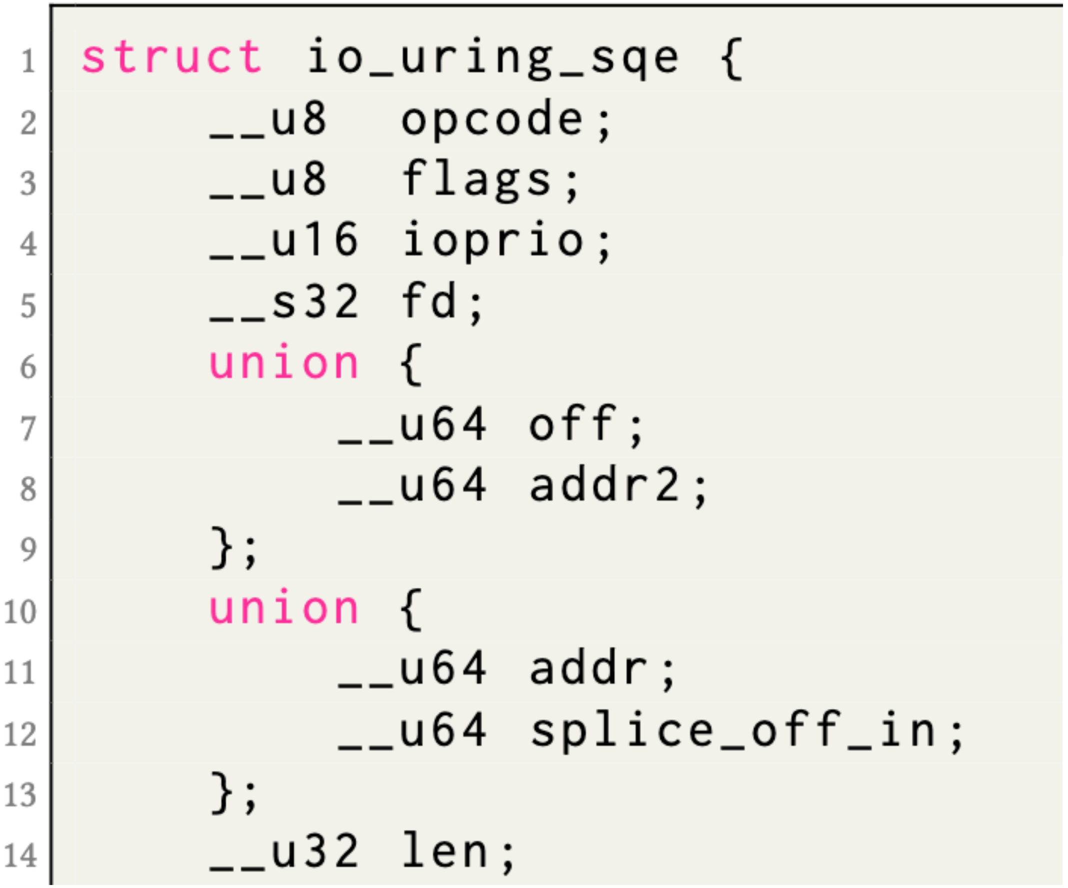
Auditing policies

- Based on the information in a single request.
- Based on multiple requests.

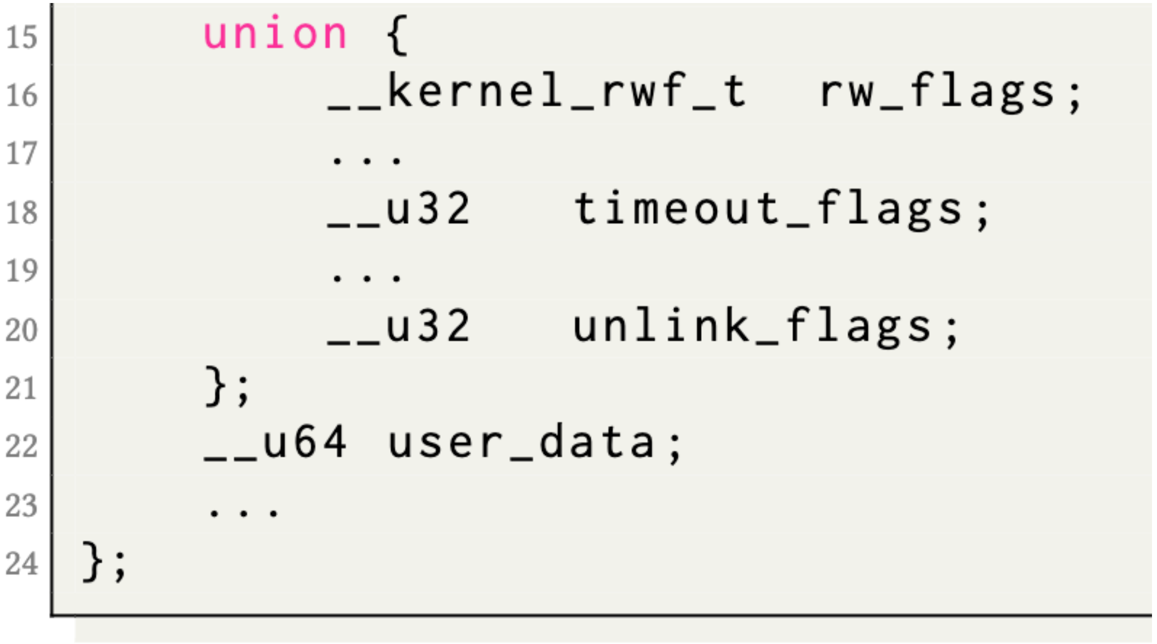
Auditing policies

- Based on the information in a single request.
- Based on multiple requests.

Lots of information in an io_uring request

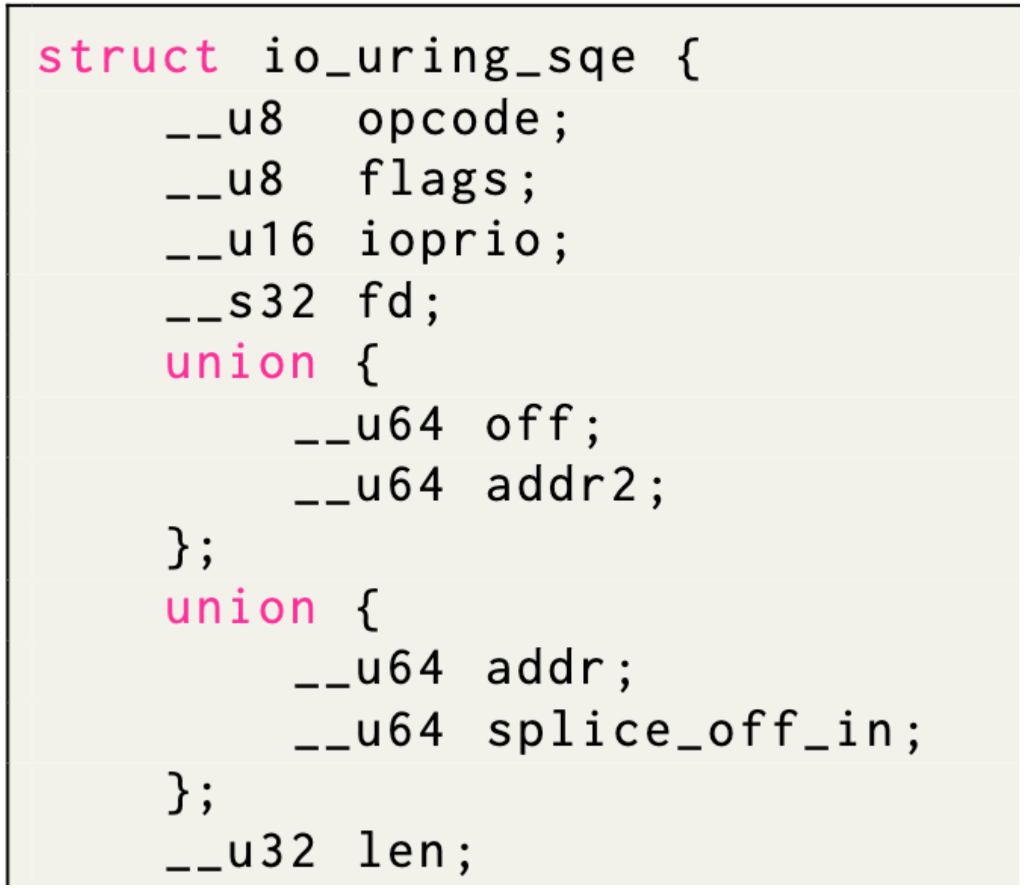


```c
struct io_uring_sqe {
    __u8  opcode;
    __u8  flags;
    __u16 ioprio;
    __s32 fd;
    union {
        __u64 off;
        __u64 addr2;
    };
    union {
        __u64 addr;
        __u64 splice_off_in;
    };
    __u32 len;
```

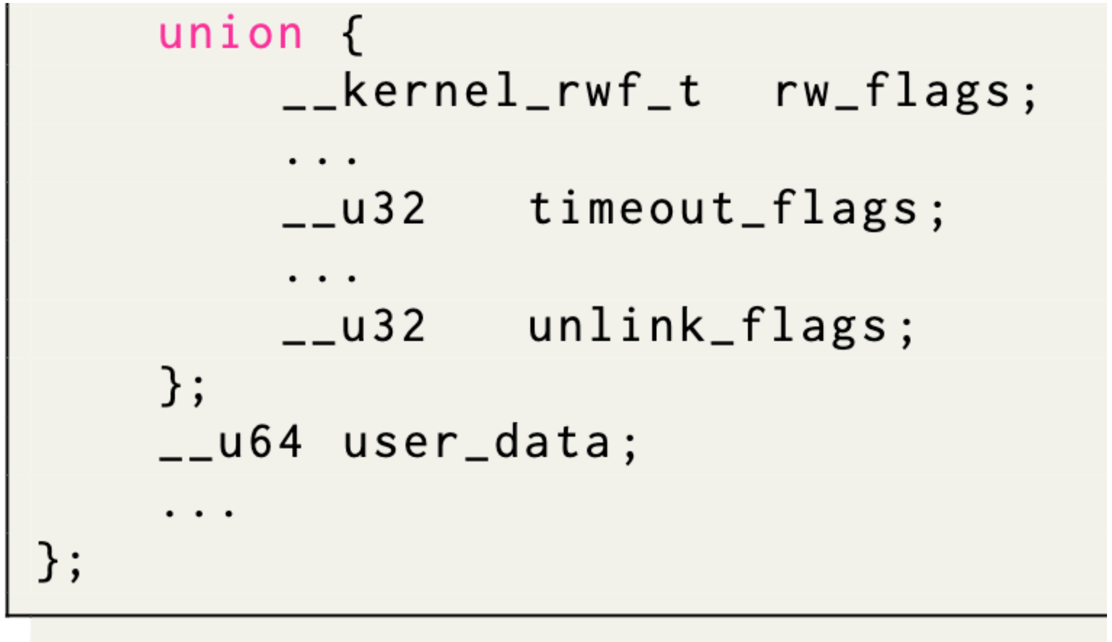


```c
    union {
        __kernel_rwf_t  rw_flags;
        ...
        __u32   timeout_flags;
        ...
        __u32   unlink_flags;
    };
    __u64 user_data;
    ...
};
```

Lots of information in an io_uring request



```
struct io_uring_sqe {
	__u8	opcode;
	__u8	flags;
	__u16 ioprio;
	__s32 fd;
	union {
		__u64 off;
		__u64 addr2;
	};
	union {
		__u64 addr;
		__u64 splice_off_in;
	};
	__u32 len;
```



```
	union {
		__kernel_rwf_t	rw_flags;
		...
		__u32	timeout_flags;
		...
		__u32	unlink_flags;
	};
	__u64 user_data;
	...
};
```

All information can be used to verify user requests!

Auditing policies

- Based on the information in a single request.
- Based on multiple requests.

Auditing policies

- Based on the information in a single request.
- Based on multiple requests.

Will give an example in the case study!

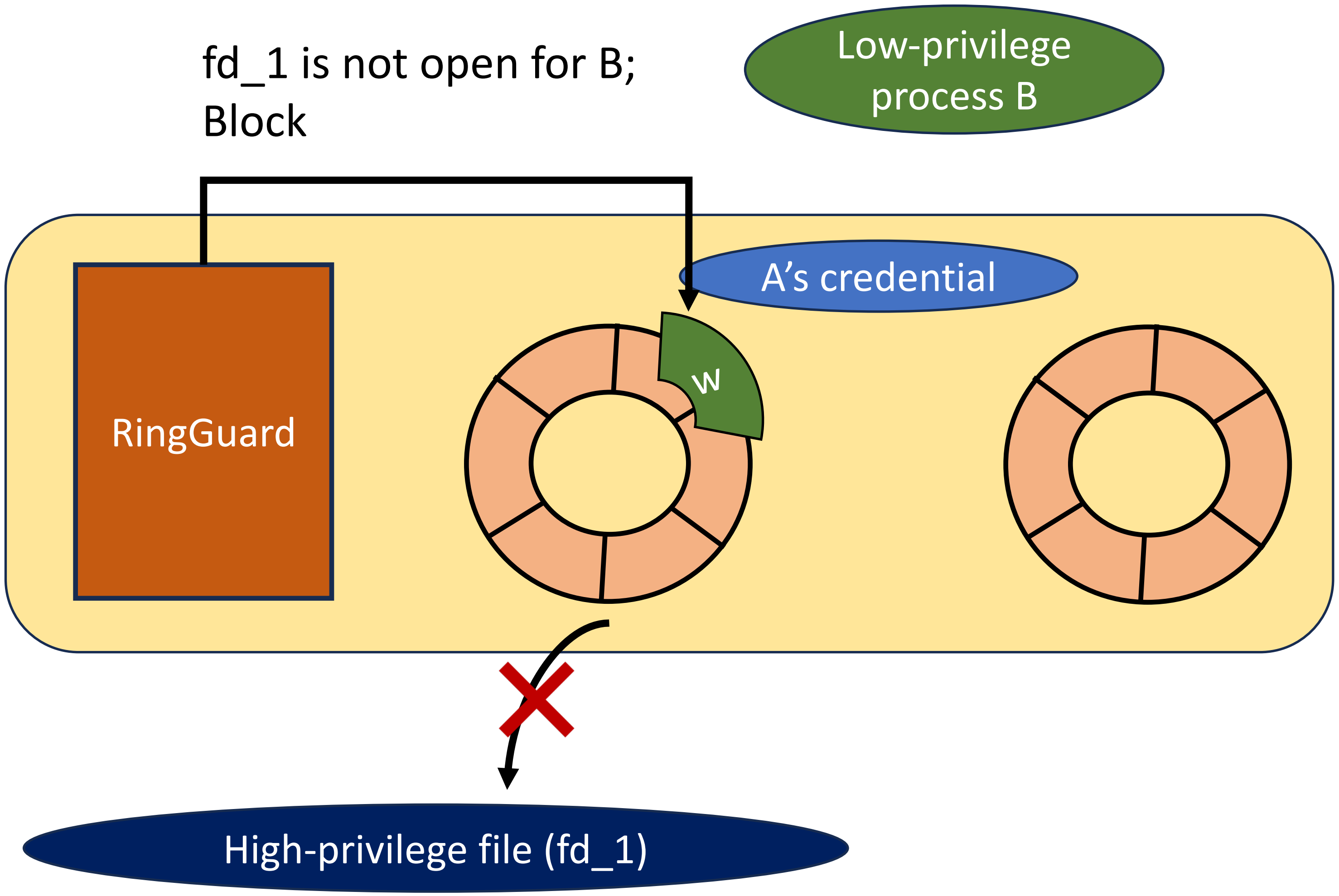
Typical use cases of RingGuard

- Sandbox the privileges of an io_uring user
- Patch io_uring vulnerabilities on the fly

Typical use cases of RingGuard

- Sandbox the privileges of an io_uring user
- Patch io_uring vulnerabilities on the fly

Sandbox process privileges with RingGuard



Sandbox process's privileges with RingGuard

- Restrict the syscalls that is able to make from an io_uring
- Impose a flexible syscall (request) filtering similar to seccomp-bpf
- Can be applied to virtual machines and containers

Typical use cases of RingGuard

- Sandbox the privileges of an io_uring user
- Patch io_uring vulnerabilities on the fly

Attack interface mitigation

- RingGuard can prevent attacks launched through io_uring requests.

CVE ID	Auditing rule
2020-29534	Check the provided file descriptor of FILES_UPDATE.
2021-3491	Check the buffer length of PROVIDE_BUFFERS.
2021-20226	Validate the existence of provided file in CLOSE.
2022-1976	Block a specific string of I/O requests.
2022-2327	Check the work flags of multiple I/O requests.
2022-4696	Check the work flags of SPLICE.
2022-29582	Block linked TIMEOUT and LINK_TIMEOUT.
2022-1508	Check multiple parameters in READ.

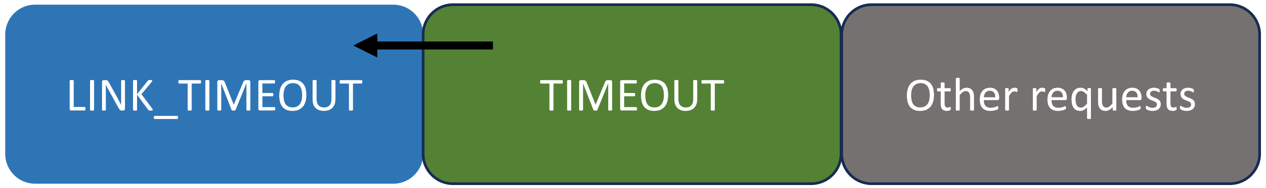
Case study: CVE-2022-29582

❑ **Related io_uring operations:**

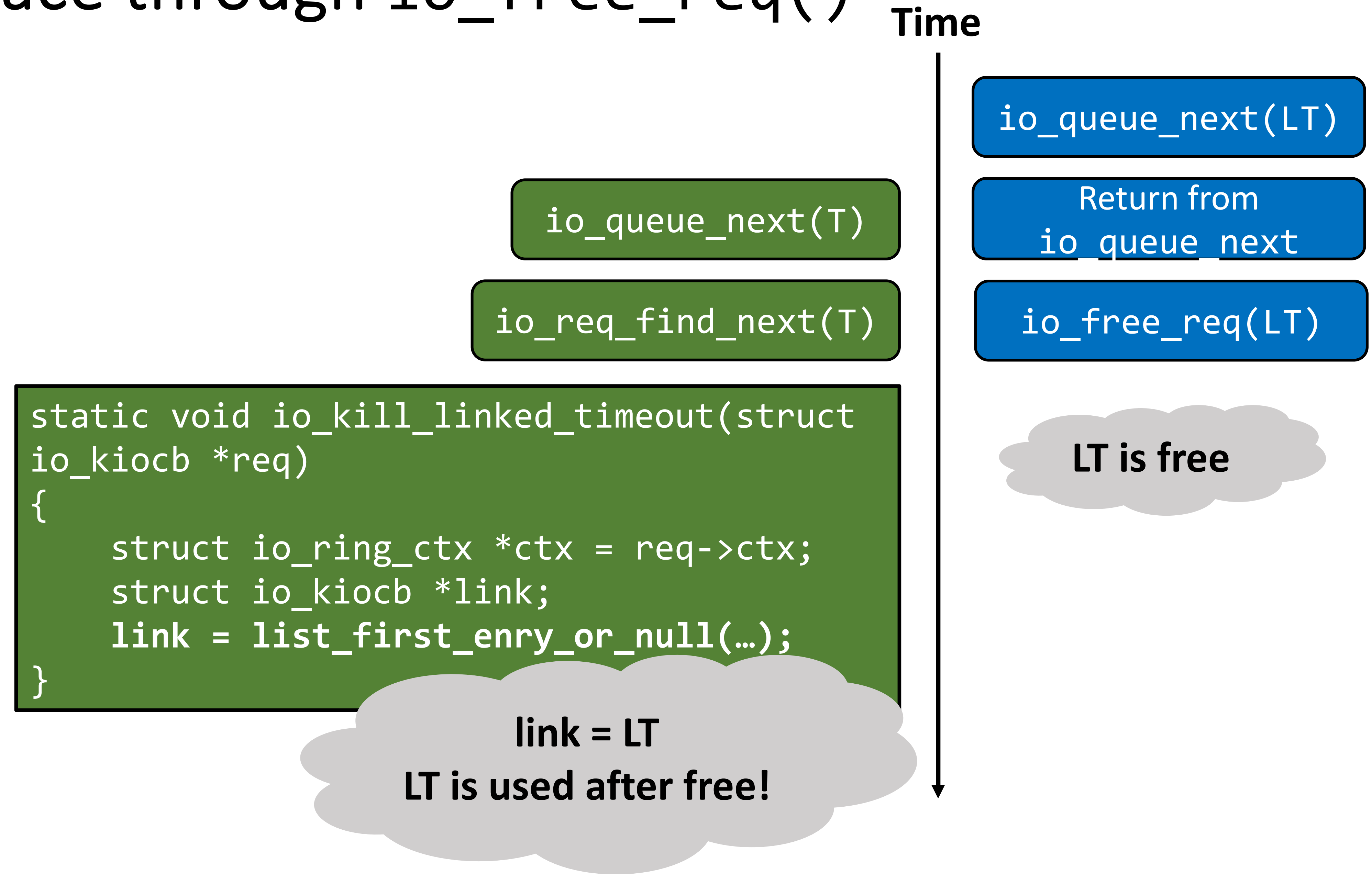
- `IORING_OP_TIMEOUT`: set a timeout event for I/O operations submitted through io_uring.
- `IORING_OP_LINK_TIMEOUT`: set a timeout event for a particular I/O operation submitted through io_uring.

❑ **Key idea:**

- Set a timeout event for `TIMEOUT` operation using `LINK_TIMEOUT` to create a *race condition* in a multicore machine, which would trigger a *use-after-free* vulnerability in the kernel.



Race through io_free_req()



Time

io_queue_next(LT)

io_queue_next(T)

Return from io_queue_next

io_req_find_next(T)

io_free_req(LT)

```
static void io_kill_linked_timeout(struct
io_kiocb *req)
{
    struct io_ring_ctx *ctx = req->ctx;
    struct io_kiocb *link;
    link = list_first_enry_or_null(…);
}
```

LT is free

link = LT
LT is used after free!

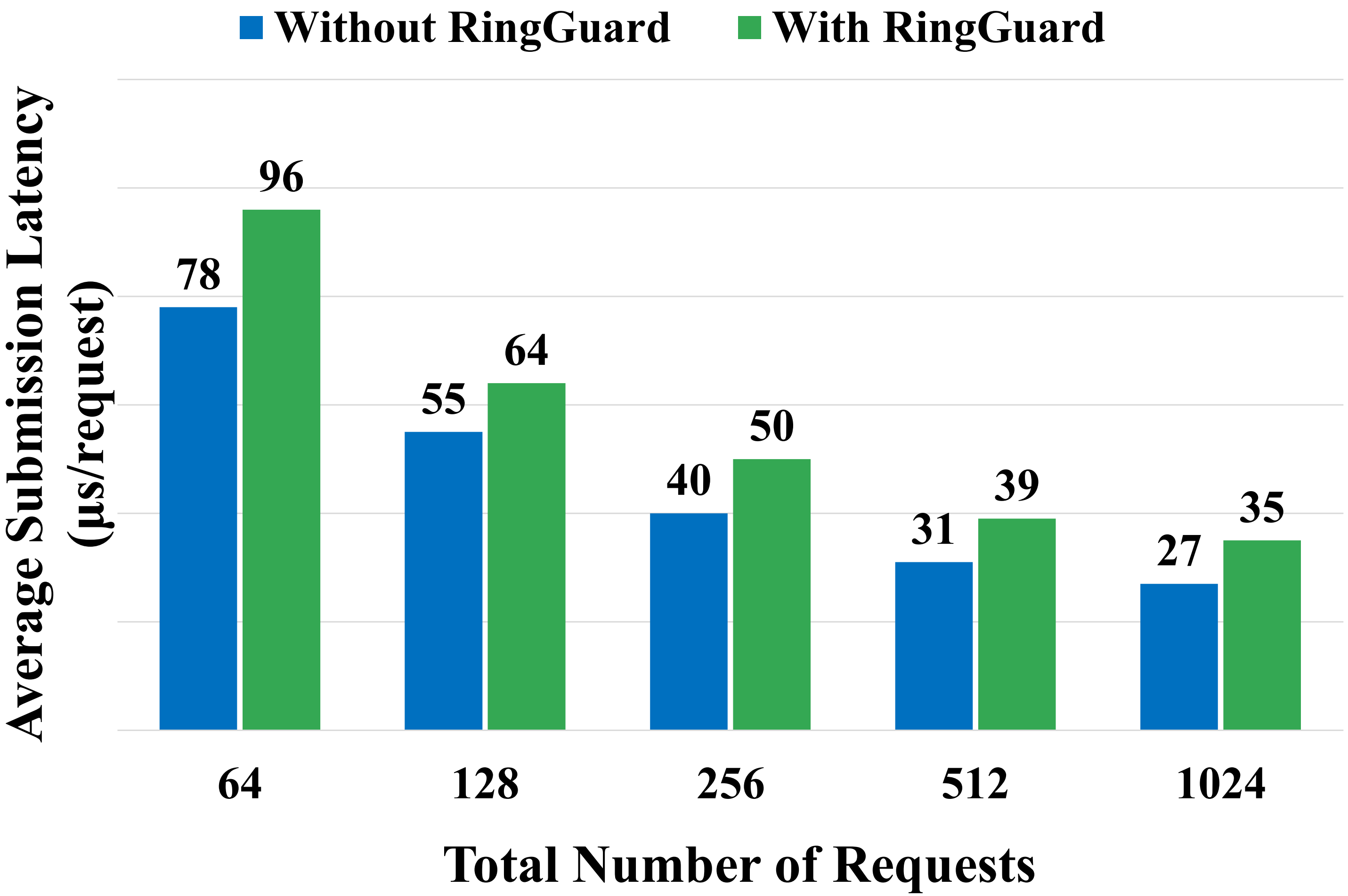
Prevent such exploit with RingGuard

Just block the linking of `TIMEOUT` and `LINK_TIMEOUT`!

❑ Rationale:

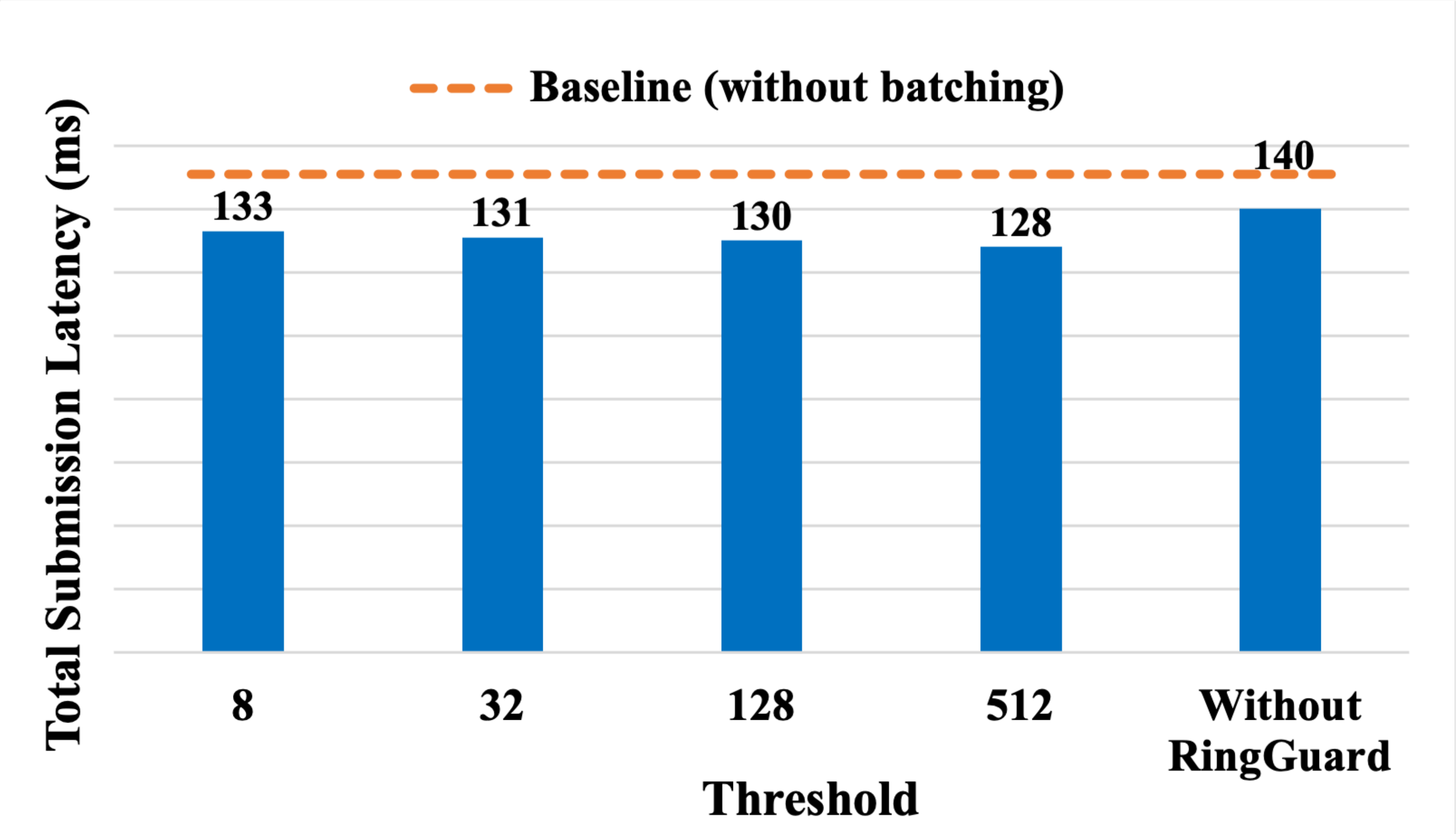
- An edge case that seldom (if any) happens
- Linked requests have some obvious features, making them easy for RingGuard to detect

RingGuard overhead



Average latency of handling NOP events with io_uring

Batch submission for better performance



Total latency of handling 512 NOP events (submitted separately) with `timeout` = 10 ms under different `threshold` values

Conclusion

- Explore the potential of combining io_uring and eBPF.
- RingGuard: A security mechanism for io_uring requests using eBPF programs.
- RingGuard imposes flexible and transparent request inspection with reasonable overhead.