black hat ASIA 2022

ExplosION: The Hidden Mines in the Android ION Driver

Le Wu, Xuen Li, Tim Xia

Baidu Security

About us

Le Wu(@Nvamous)

- Focus on Android/Linux bug hunting and exploit
- Found 200+ vulnerabilities in the last two years
- Top1 in Android Chipset Security Program, Top1 in MediaTek Mobile Security Program

Xuen Li(@lxn524)

- Interested in Android, Linux Kernel and OSS security testing and exploitation
- Focus on PSA and OP-TEE currently
- Author of One Click Root Master, SuperRootMaster, etc…

Tim Xia

- Staff security researcher at Baidu Inc.
- Focused on system and software security solutions
- Previous PacSec and HITB speaker

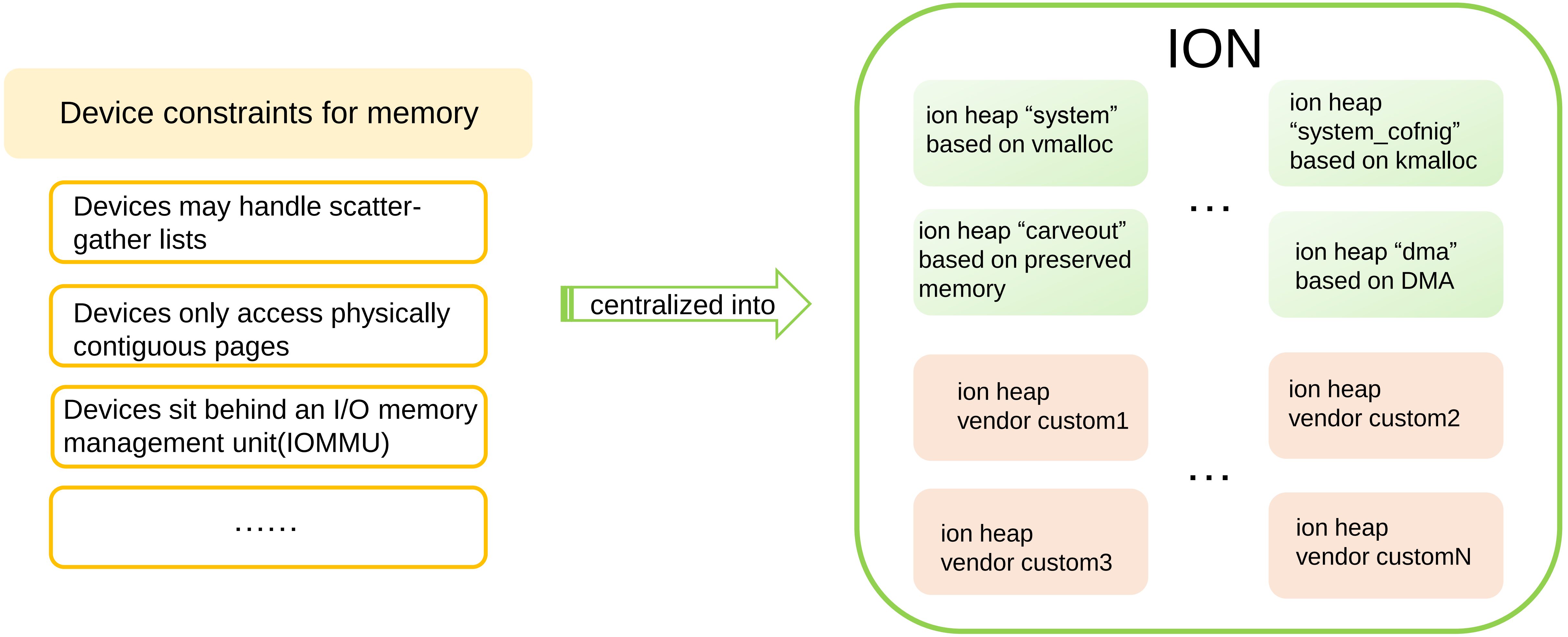
Agenda

- Introduction to ION
- Using ION
- Diving into ExplosION
- Reflections on ExplosION
- Future work

Introduction——What is ION?

- A generalized memory pool manager

  It’s used to address the issue of fragmented memory management interfaces across different Android devices.


Device constraints for memory
Devices may handle scatter-gather lists
Devices only access physically contiguous pages
Devices sit behind an I/O memory management unit(IOMMU)
……
centralized into
ION
ion heap “system” based on vmalloc
ion heap “system_cofnig” based on kmalloc
…
ion heap “carveout” based on preserved memory
ion heap “dma” based on DMA
ion heap vendor custom1
ion heap vendor custom2
…
ion heap vendor custom3
ion heap vendor customN

Introduction——Why ION?

- A common Android driver used for a decade
  - ✓ Although ION is replaced with dma-buf in Android 12, millions of devices still use ION
  - ✓ ION will survive for a long time due to fragmentation of Android
- Can be accessed by untrusted apps
  - ✓ No additional privileges needed to trigger the vulnerabilities in ION
  - ✓ Perfect exploitation target, like the BINDER device

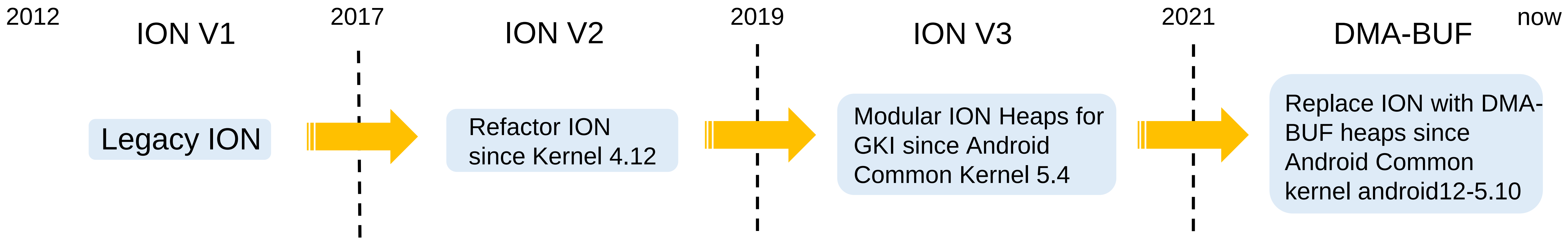
Introduction——Why ION?

- A memory management component

  Vulnerabilities in memory management components are proven to be dangerous!

DIRTY COW

- Multiple refactors from upstream



2012 ION V1: Legacy ION

2017 ION V2: Refactor ION since Kernel 4.12

2019 ION V3: Modular ION Heaps for GKI since Android Common Kernel 5.4

2021 DMA-BUF (now): Replace ION with DMA-BUF heaps since Android Common kernel android12-5.10

Why there are multiple refactors? Why ION ends up with a removal in Android kernel?

Maybe there are security issues in it?

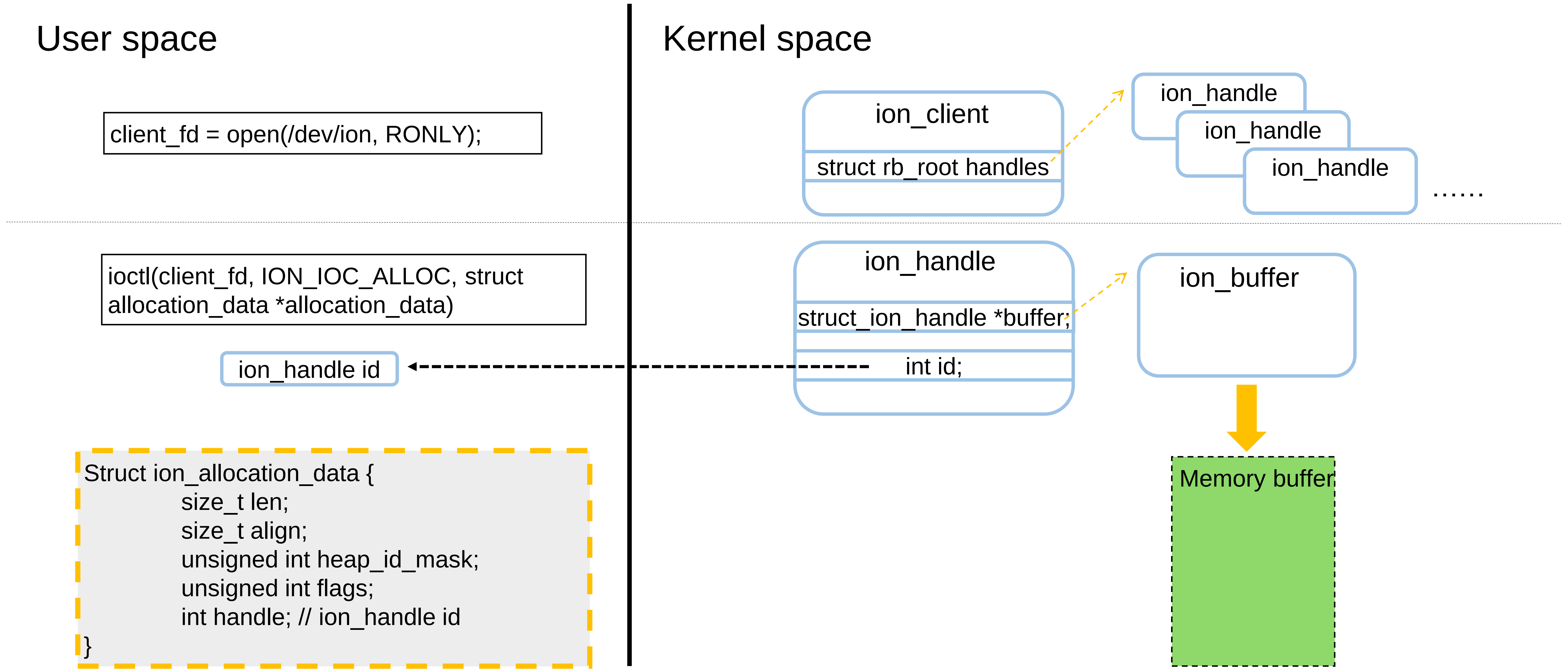
So far, we have found 40+ vulnerabilities in ION, and millions of devices are affected!

Before diving into ExplosION, let’s have a look at the using of legacy ION

Why legacy ION?
It shows us the most complete design and implementation of ION. Subsequent refactors are all based on it.

Using ION

- Allocate memory buffer from user space



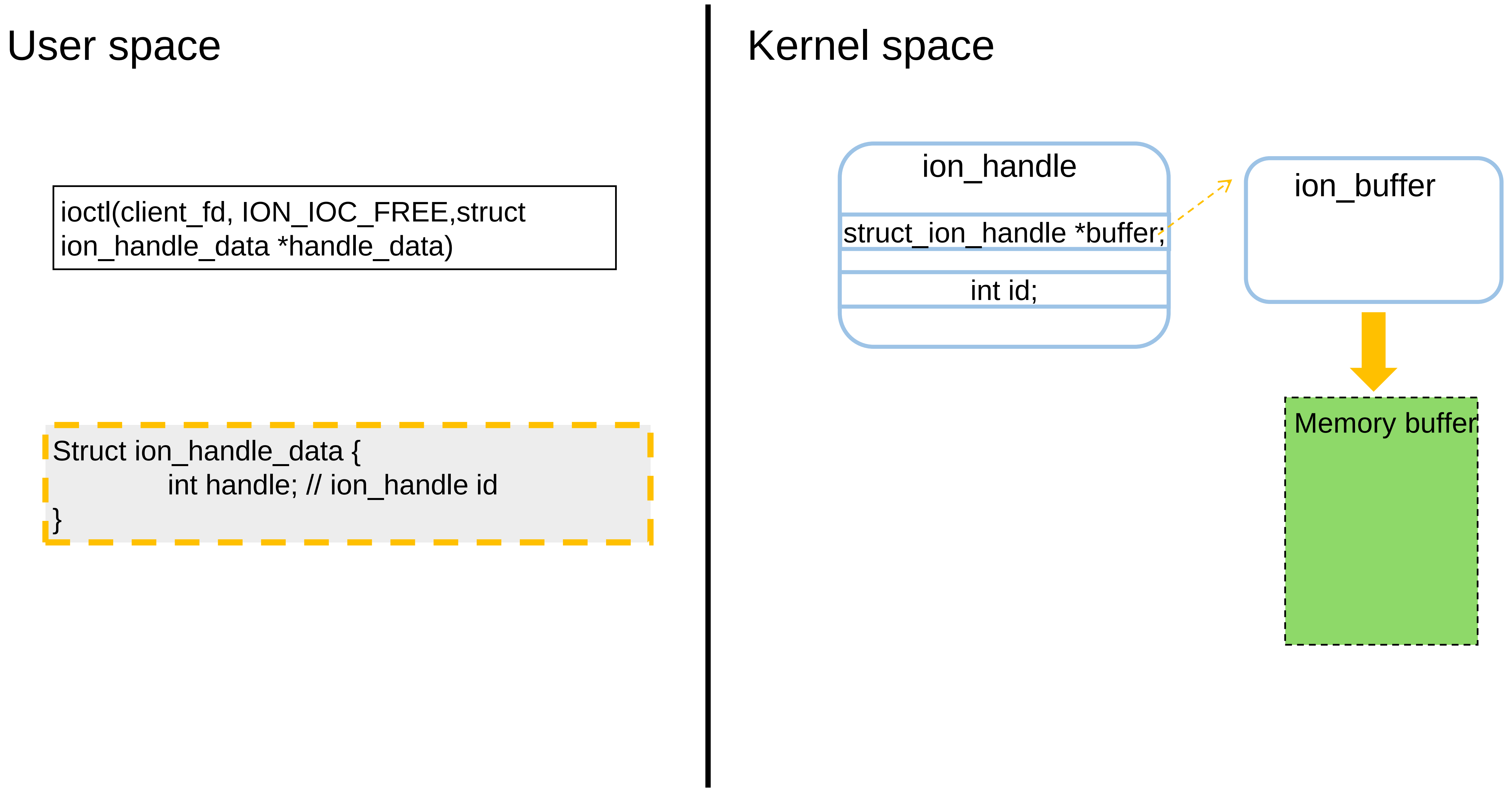
```
Struct ion_allocation_data {
        size_t len;
        size_t align;
        unsigned int heap_id_mask;
        unsigned int flags;
        int handle; // ion_handle id
}
```

Using ION

- Free memory buffer from user space

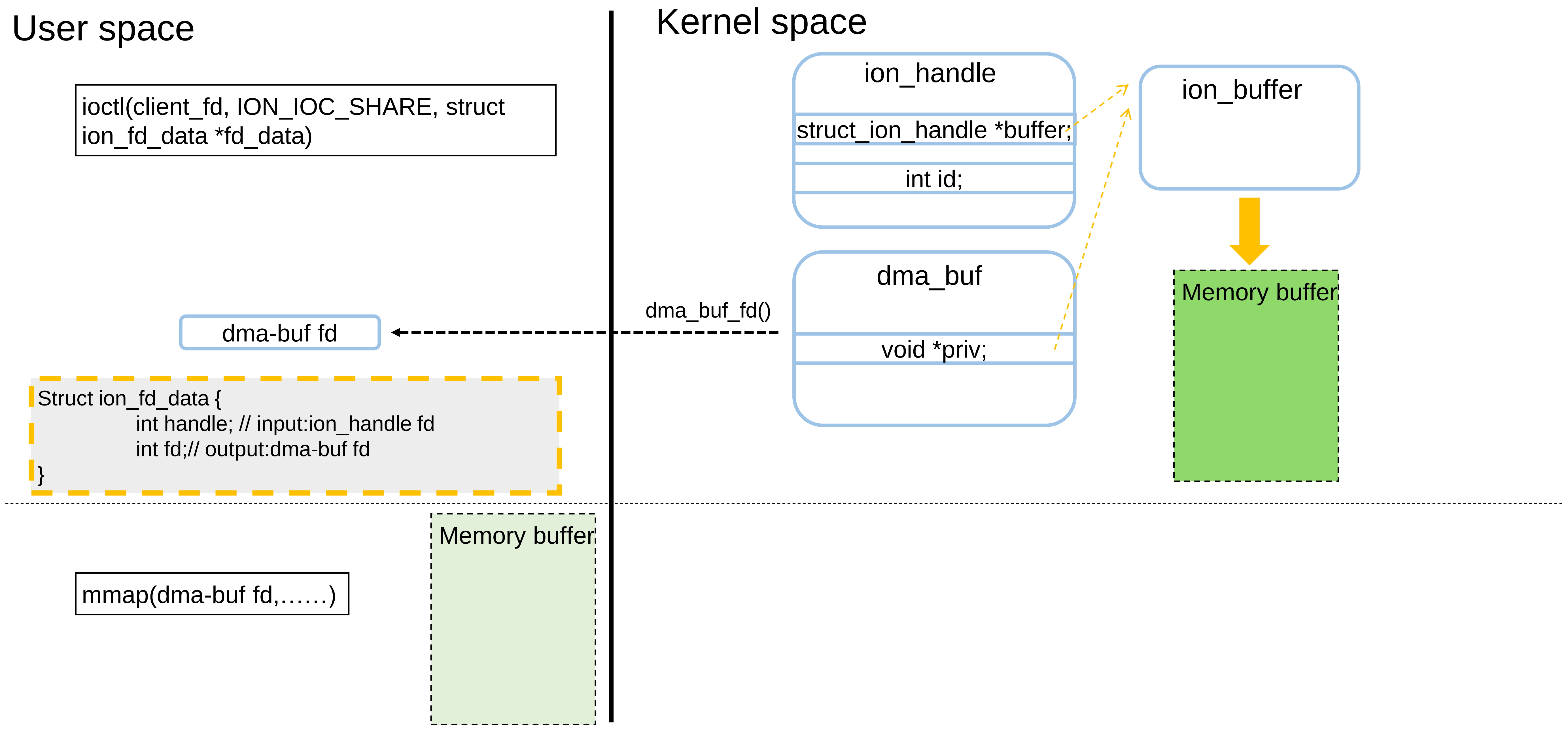
User space

Kernel space


ioctl(client_fd, ION_IOC_FREE,struct ion_handle_data *handle_data)
Struct ion_handle_data {
int handle; // ion_handle id
}
ion_handle
struct_ion_handle *buffer;
int id;
ion_buffer
Memory buffer

Using ION

- Buffer sharing from user space



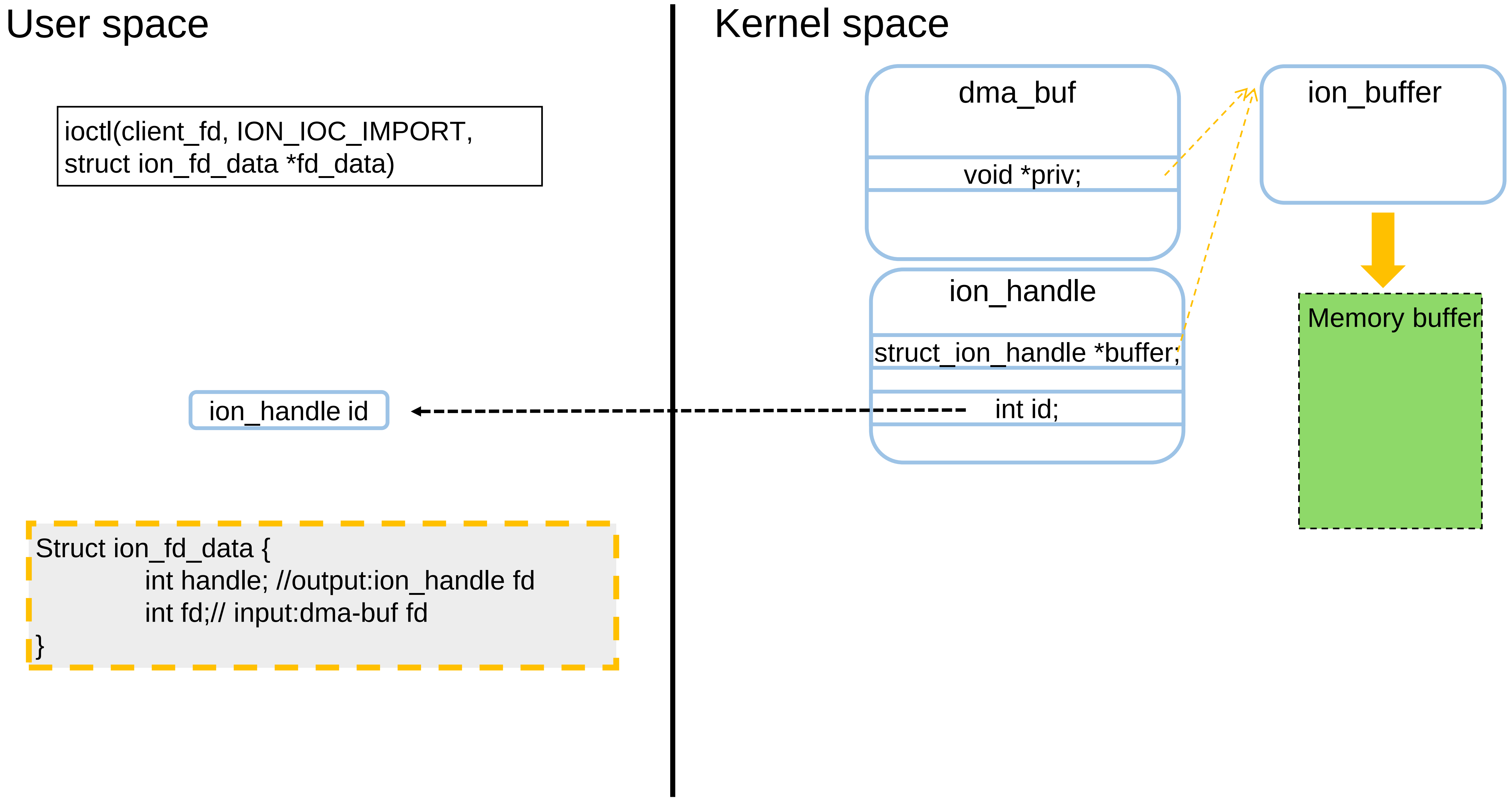
```
ioctl(client_fd, ION_IOC_SHARE, struct
ion_fd_data *fd_data)
```

```
Struct ion_fd_data {
              int handle; // input:ion_handle fd
              int fd;// output:dma-buf fd
}
```

```
mmap(dma-buf fd,……)
```

Using ION

- Import buffer from user space



```
Struct ion_fd_data {
        int handle; //output:ion_handle fd
        int fd;// input:dma-buf fd
}
```

Using ION

- Allocate memory buffer from kernel space

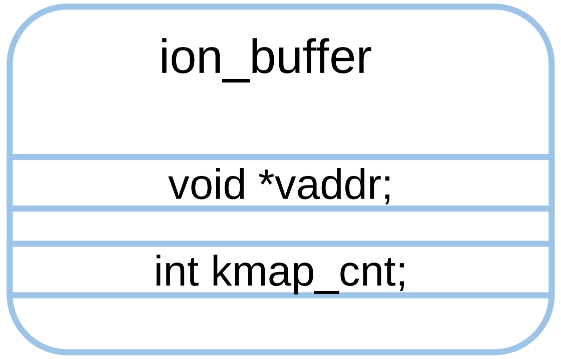
```
struct ion_handle *ion_alloc(struct ion_client *client, size_t len,
                             size_t align, unsigned int heap_id_mask,
                             unsigned int flags);
```

- Free memory buffer from kernel space

```
void ion_free(struct ion_client *client,
              struct ion_handle *handle);
```

- Map the memory buffer into kernel space

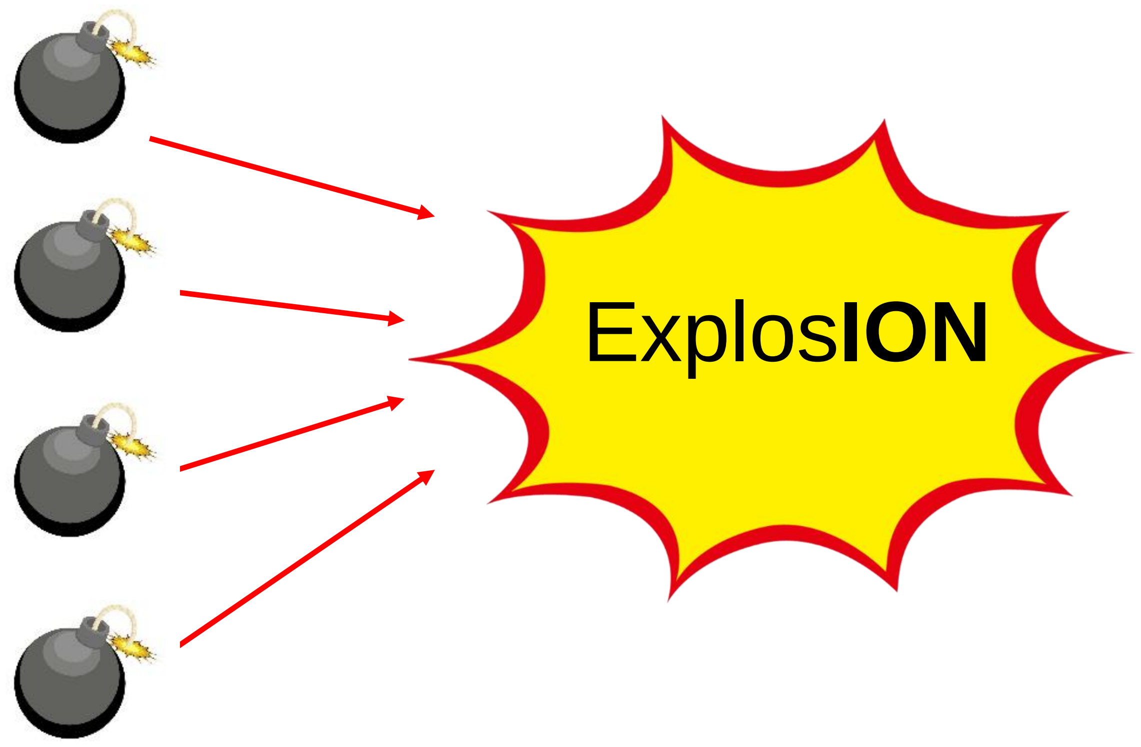
```
void *ion_map_kernel(struct ion_client *client, struct ion_handle *handle);
```



Diving into ExplosION

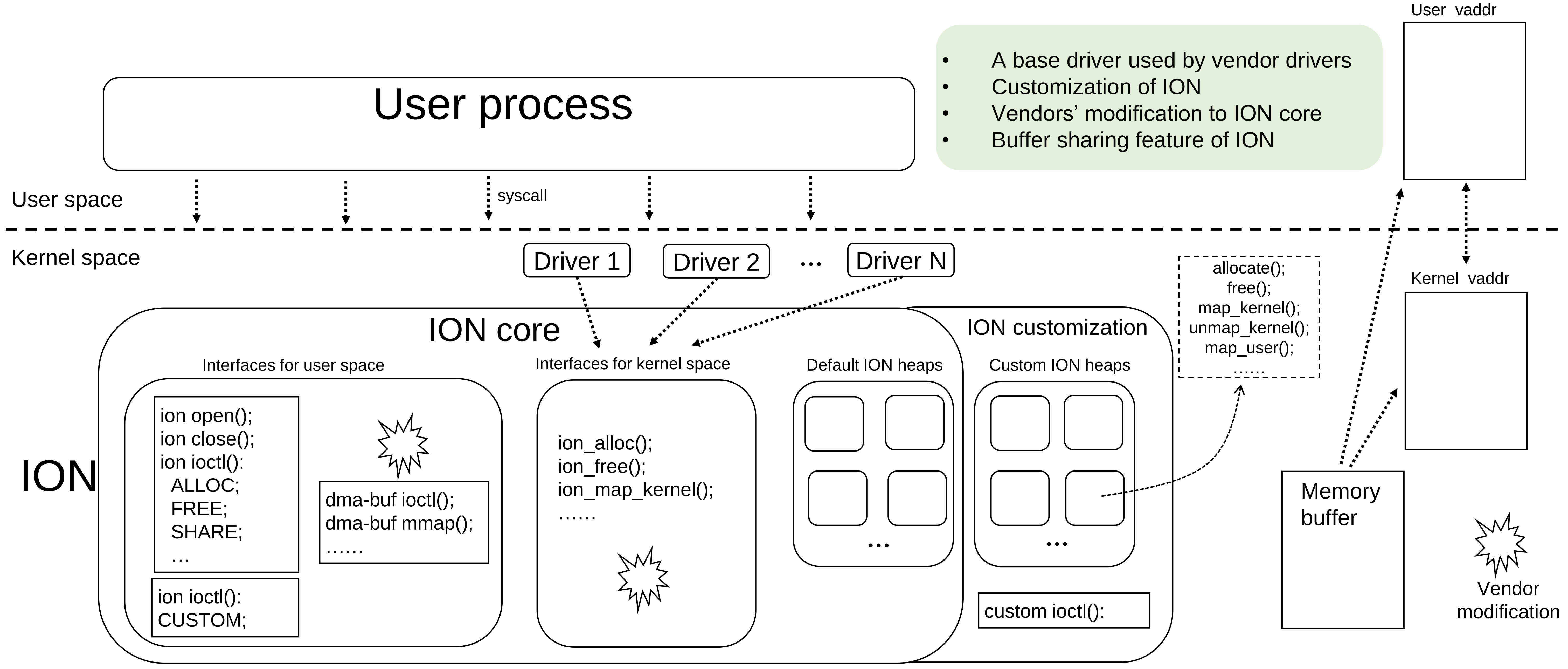
The Characteristics of ION:

- A base driver used by vendor drivers
- Customization of ION
- Vendors’ modification to ION core
- Buffer sharing feature of ION

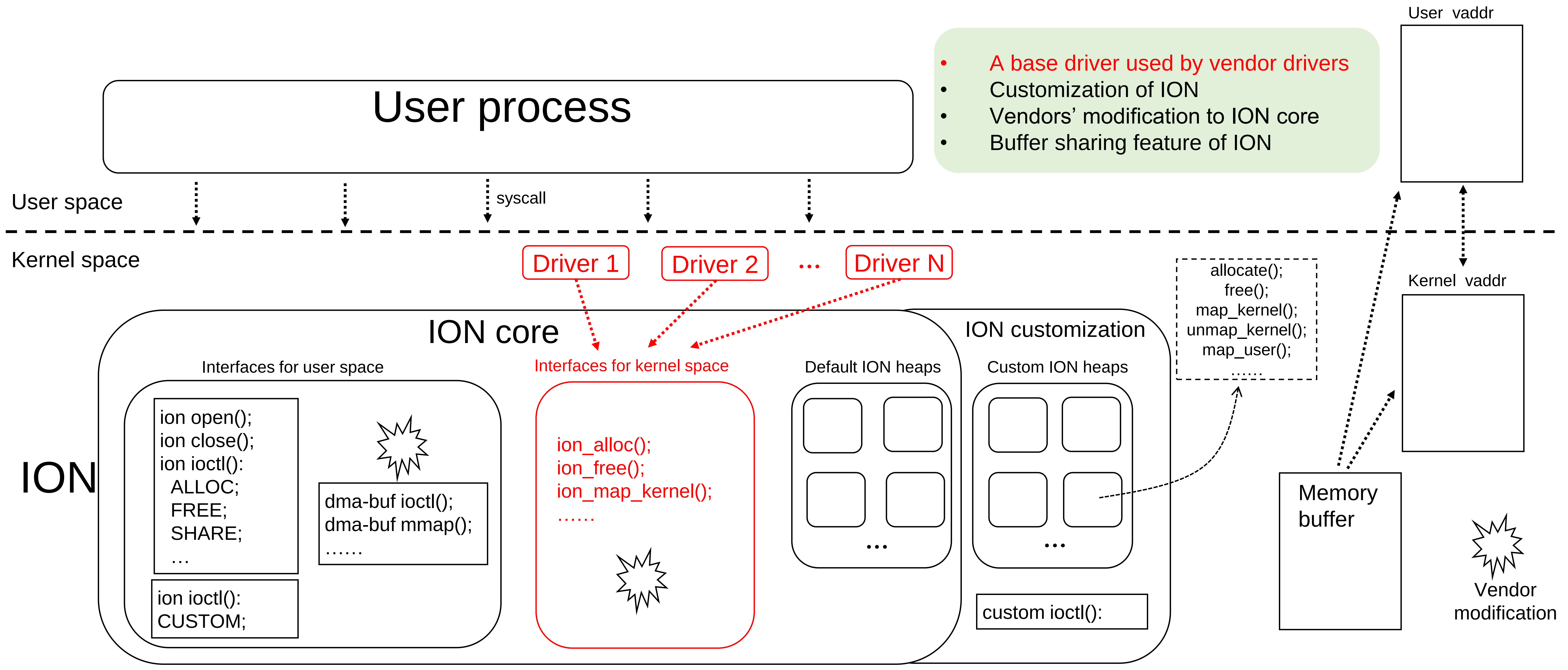


Legacy ION Architecture

- A base driver used by vendor drivers
- Customization of ION
- Vendors' modification to ION core
- Buffer sharing feature of ION



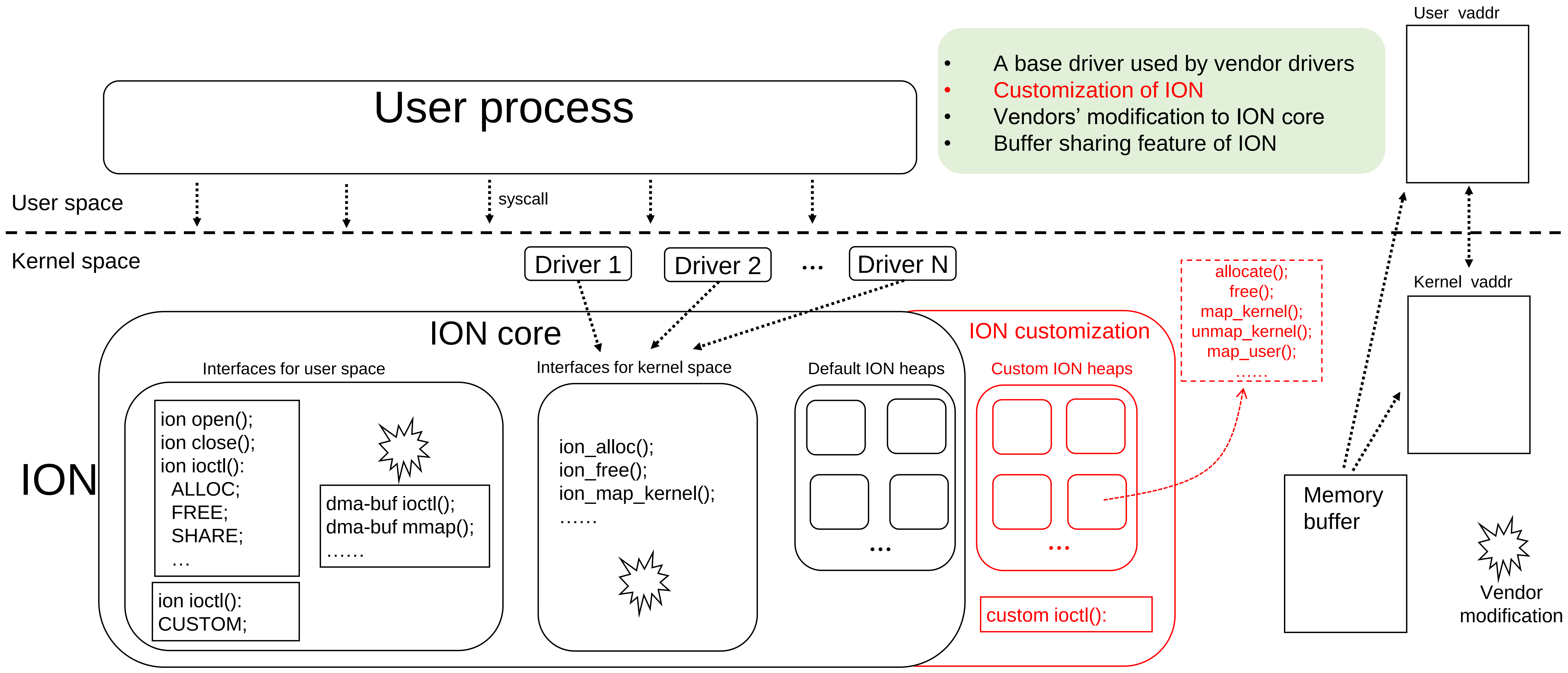
Legacy ION Architecture



- A base driver used by vendor drivers
- Customization of ION
- Vendors’ modification to ION core
- Buffer sharing feature of ION

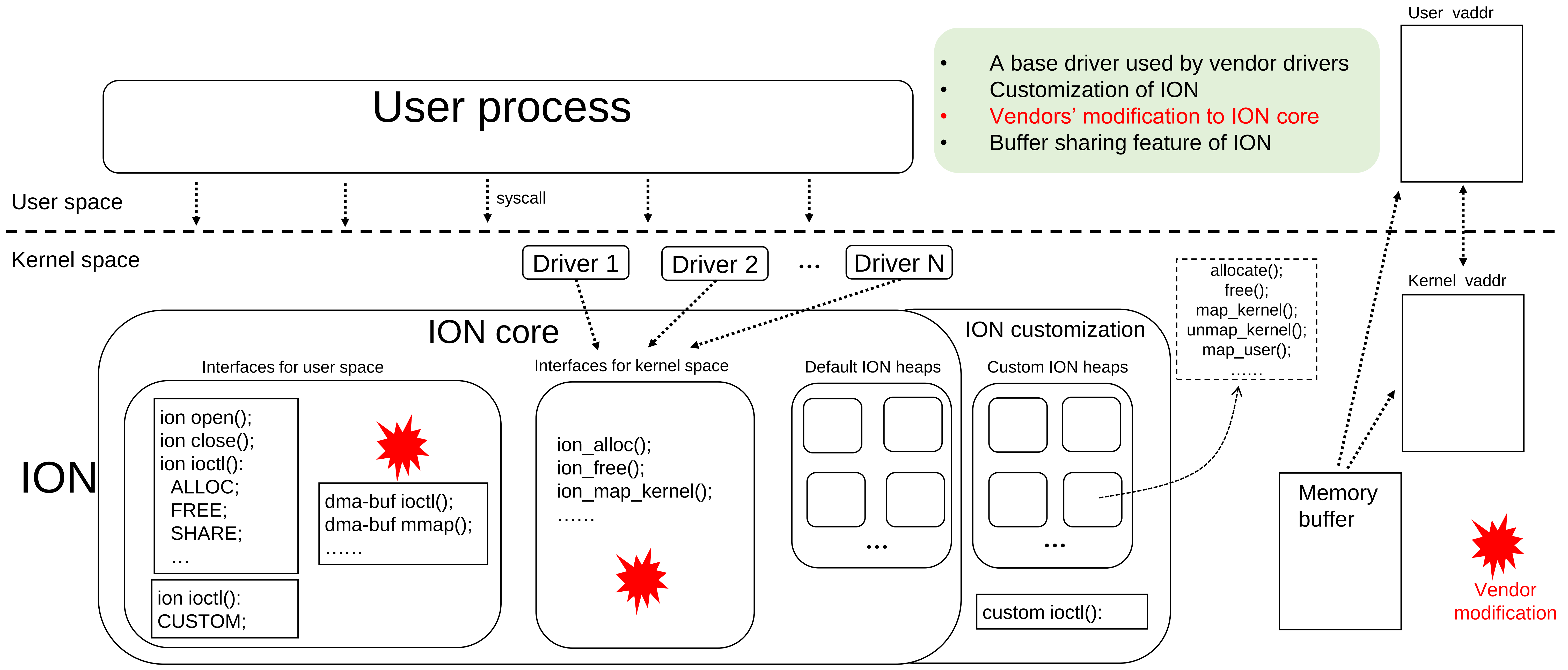
Legacy ION Architecture

- A base driver used by vendor drivers
- Customization of ION
- Vendors' modification to ION core
- Buffer sharing feature of ION

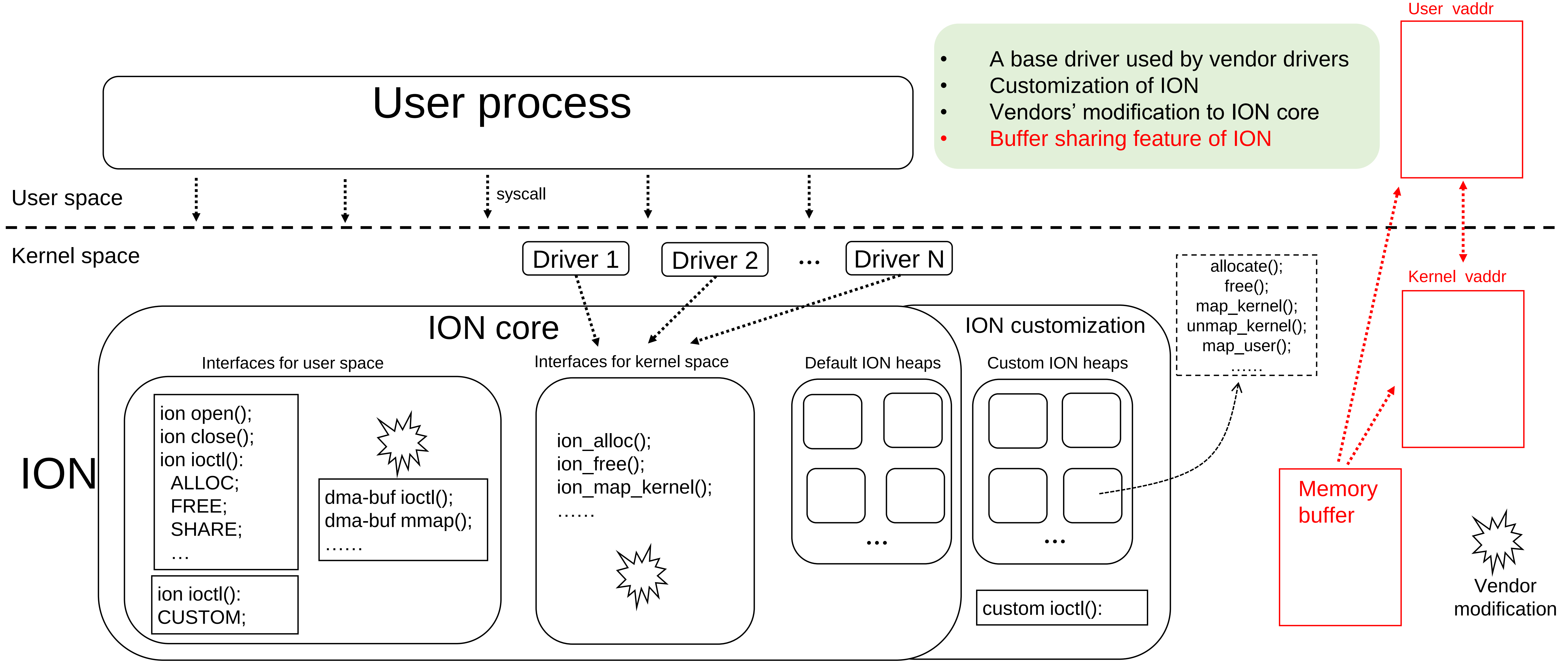


Legacy ION Architecture

- A base driver used by vendor drivers
- Customization of ION
- Vendors' modification to ION core
- Buffer sharing feature of ION

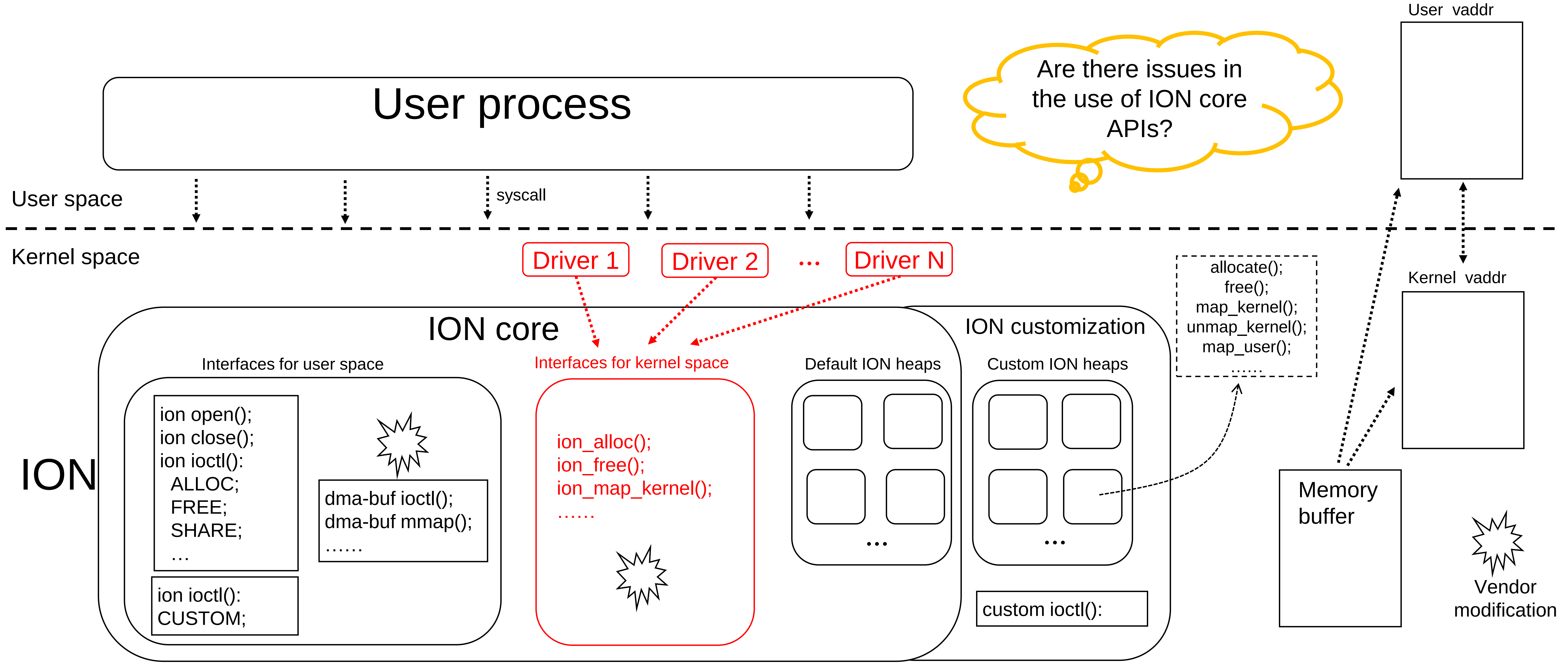


Legacy ION Architecture

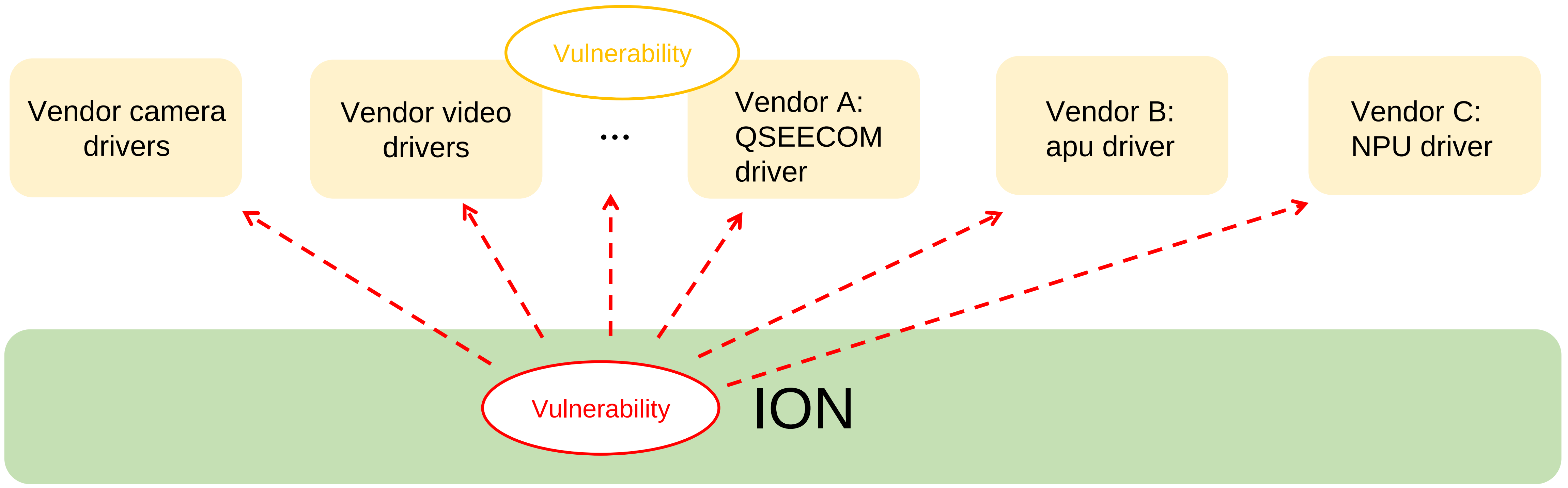


- A base driver used by vendor drivers
- Customization of ION
- Vendors' modification to ION core
- Buffer sharing feature of ION

A base driver used by vendor drivers



A base driver used by vendor drivers



Vulnerability： From ION Core
Vulnerability： Misuse of ION Core APIs

A base driver used by vendor drivers

- Analyze the design and implementation of ION core APIs for kernel space

- Check if vendor drivers use ION core APIs correctly

- Vulnerabilities from ION core：

  CVE-2021-0929

  CVE-2021-39714

- Misuse of ION Core APIs by vendor driver：

  CVE-2022-20110

CVE-2021-0929

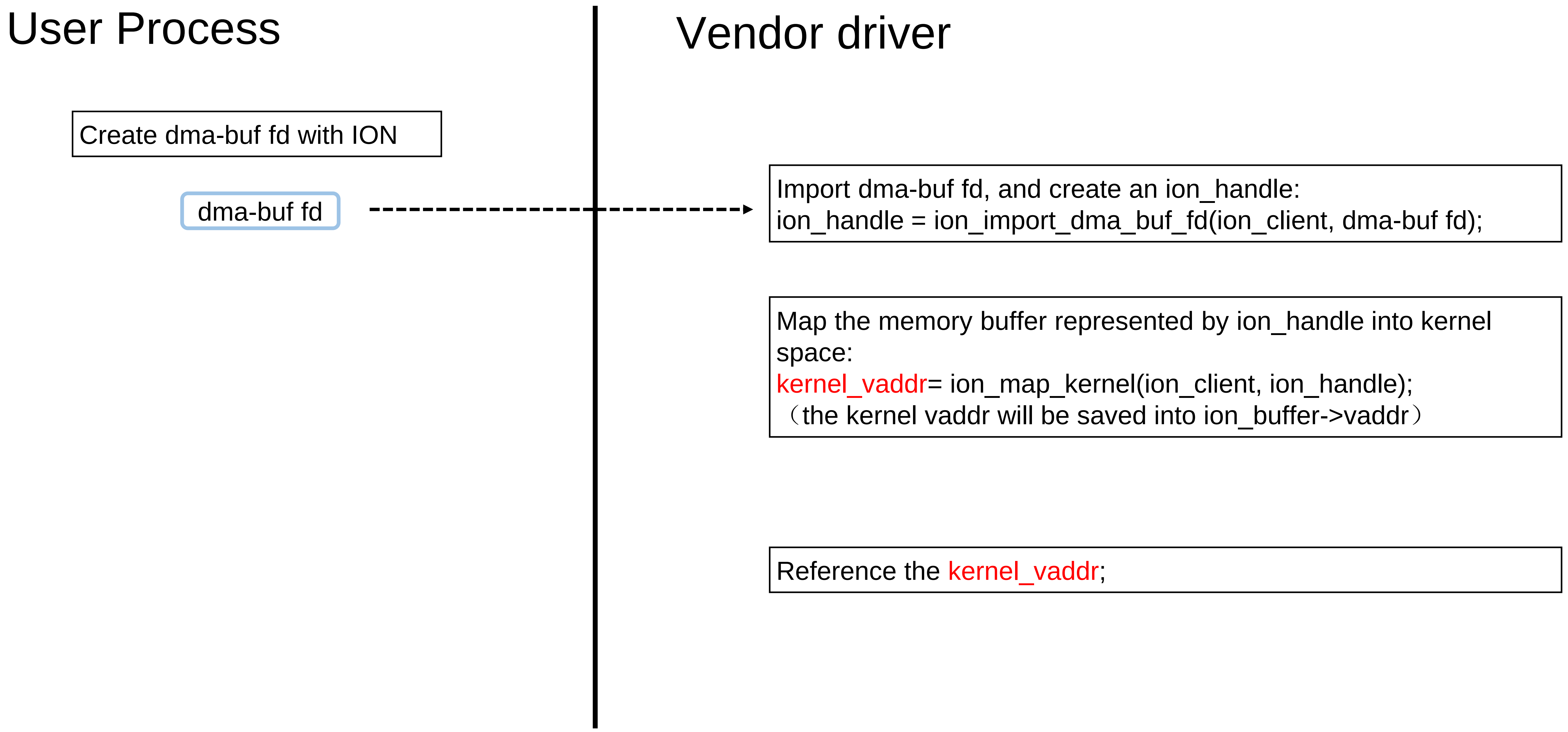
- Affected ION version:
  ION V1,V2,V3

- Root cause:
  ION core exposed an API to user space which it shouldn’t be

- Impact:
  An UAF issue would be introduced into vendor drivers by ION core in a specific use scenario

CVE-2021-0929

An ION use scenario



User Process

Vendor driver

Create dma-buf fd with ION

dma-buf fd

Import dma-buf fd, and create an ion_handle:
ion_handle = ion_import_dma_buf_fd(ion_client, dma-buf fd);

Map the memory buffer represented by ion_handle into kernel space:
kernel_vaddr= ion_map_kernel(ion_client, ion_handle);
（the kernel vaddr will be saved into ion_buffer->vaddr）

Reference the kernel_vaddr;

CVE-2021-0929

```
static long dma_buf_ioctl(struct file *file,
                                  unsigned int cmd, unsigned long arg)
{
        ……
        dmabuf = file->private_data;
        switch (cmd) {
        case DMA_BUF_IOCTL_SYNC:
                ……
                if (sync.flags & DMA_BUF_SYNC_END)
                        ret = dma_buf_end_cpu_access(dmabuf,
direction);
                ……
```

```
int dma_buf_end_cpu_access(struct dma_buf *dmabuf,
                                  enum dma_data_direction direction)
{
        int ret = 0;
        WARN_ON(!dmabuf);
        if (dmabuf->ops->end_cpu_access)
                ret = dmabuf->ops->end_cpu_access(dmabuf, direction);
        return ret;
}
```

```
static int ion_dma_buf_end_cpu_access(struct dma_buf *dmabuf,
                                                  …)
{
        struct ion_buffer *buffer = dmabuf->priv;
        mutex_lock(&buffer->lock);
        ion_buffer_kmap_put(buffer);
        ……
}
```

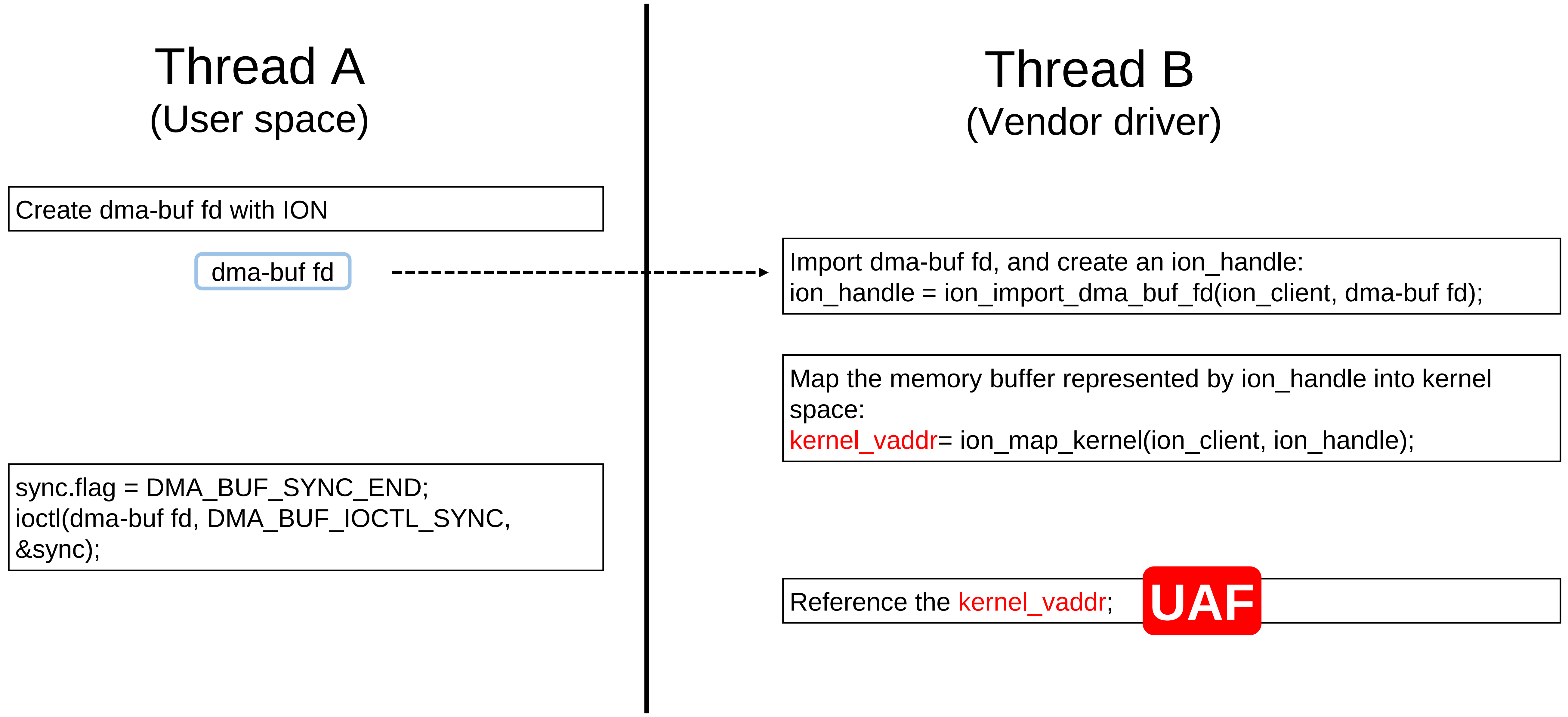
……

```
void ion_heap_unmap_kernel(struct ion_heap *heap,
                                 struct ion_buffer *buffer)
{
        vunmap(buffer->vaddr);
}
```

buffer->vaddr gets unmmaped in kernel space!!!

CVE-2021-0929

UAF would happen in a race condition:



Thread A
(User space)

Create dma-buf fd with ION

dma-buf fd

```
sync.flag = DMA_BUF_SYNC_END;
ioctl(dma-buf fd, DMA_BUF_IOCTL_SYNC,
&sync);
```

Thread B
(Vendor driver)

Import dma-buf fd, and create an ion_handle:
```
ion_handle = ion_import_dma_buf_fd(ion_client, dma-buf fd);
```

Map the memory buffer represented by ion_handle into kernel space:
```
kernel_vaddr= ion_map_kernel(ion_client, ion_handle);
```

Reference the kernel_vaddr;

CVE-2022-20110

- Affected ION version:
  A vendor’s devices with ION V1
- Root cause:
  Misuse of ION core APIs by vendor driver
- Impact:
  UAF

CVE-2022-20110

```
void *ion_map_kernel(struct ion_client *client, struct
ion_handle *handle)
{
        struct ion_buffer *buffer;
        void *vaddr;

        mutex_lock(&client->lock);
        ……
        buffer = handle->buffer;
        ……
        mutex_lock(&buffer->lock);
        vaddr = ion_handle_kmap_get(handle);
        mutex_unlock(&buffer->lock);
        mutex_unlock(&client->lock);
        return vaddr;
}
```

……

```
static void *ion_buffer_kmap_get(struct ion_buffer *buffer)
{
        void *vaddr;

        if (buffer->kmap_cnt) {
                buffer->kmap_cnt++;
                return buffer->vaddr;
        }
        vaddr = buffer->heap->ops->map_kernel(buffer->heap,
buffer);
        ……
        buffer->vaddr = vaddr;
        buffer->kmap_cnt++;
        return vaddr;
}
```

buffer->vaddr and buffer->kmap_cnt are protected by mutex locks!

CVE-2022-20110

Access internal data instead of using the exported API:

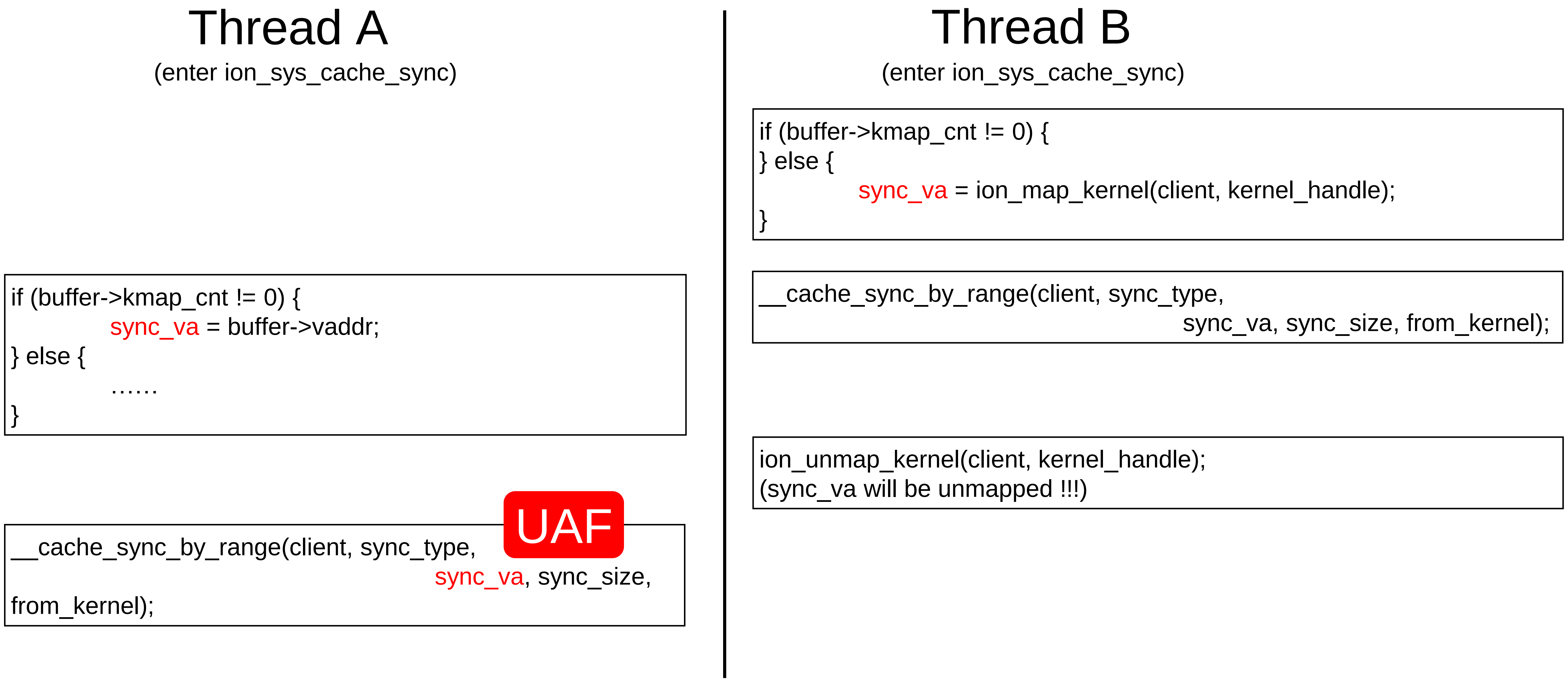
```
static long ion_sys_cache_sync(struct ion_client *client,
                               struct ion_sys_cache_sync_param *param,
                               int from_kernel)
{
       ……
       unsigned long sync_va = 0;
       struct ion_buffer *buffer;
                     ……
                     if (buffer->kmap_cnt != 0) {
                            sync_va = (unsigned long)buffer->vaddr;
                     } else {
                            sync_va = (unsigned long)ion_map_kernel(client, kernel_handle);
                            ion_need_unmap_flag = 1;
                     }
       ……
       ret = __cache_sync_by_range(client, sync_type,
                                   sync_va, …);
              ……
              ion_unmap_kernel(client, kernel_handle);
……
```

buffer->vaddr and buffer->kmap_cnt are not protected by mutex locks!

UAF

CVE-2022-20110

UAF would happen in a race condition:



Thread A

(enter ion_sys_cache_sync)

```
if (buffer->kmap_cnt != 0) {
        sync_va = buffer->vaddr;
} else {
        ......
}
```

```
__cache_sync_by_range(client, sync_type,
                                    sync_va, sync_size,
from_kernel);
```

Thread B

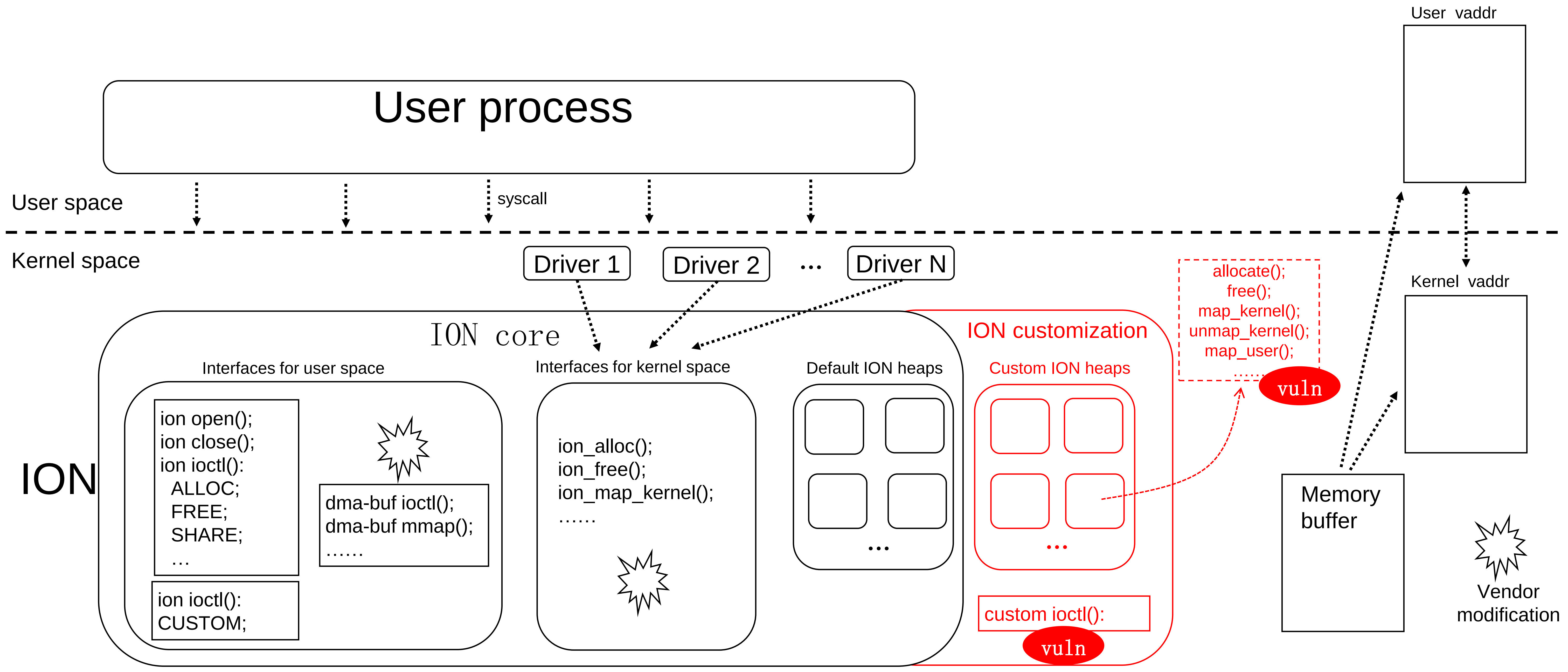
(enter ion_sys_cache_sync)

```
if (buffer->kmap_cnt != 0) {
} else {
        sync_va = ion_map_kernel(client, kernel_handle);
}
```

```
__cache_sync_by_range(client, sync_type,
                                    sync_va, sync_size, from_kernel);
```

```
ion_unmap_kernel(client, kernel_handle);
(sync_va will be unmapped !!!)
```

Customization of ION



Customization of ION

✓ To handle all the various types of hardware memory allocation, ION provides interfaces for vendors to add their own ION heaps(except common heaps)

```
        void ion_device_add_heap(struct ion_device *dev, struct ion_heap *heap);
```

```
Struct ion_heap {
……
Struct ion_device *dev;
Emum ion_heap_type type;
Struct ion_heap_ops *ops;
Unsigned long flags;
Unsinged int id;
……
}
```

```
struct ion_heap_ops {
        int (*allocate)(struct ion_heap *heap,
                        struct ion_buffer *buffer, unsigned long len,
                        unsigned long align, unsigned long flags);
        void (*free)(struct ion_buffer *buffer);
        void * (*map_kernel)(struct ion_heap *heap, struct ion_buffer *buffer);
        void (*unmap_kernel)(struct ion_heap *heap, struct ion_buffer *buffer);
        int (*map_user)(struct ion_heap *mapper, struct ion_buffer *buffer,
                        struct vm_area_struct *vma);
        int (*shrink)(struct ion_heap *heap, gfp_t gfp_mask, int nr_to_scan);
};
```

✓ ION provides an ION_IOC_CUSTOM ioctl command which allows vendors to implement their own buffer operations

```
case ION_IOC_CUSTOM:
{
        if (!dev->custom_ioctl)
                return -ENOTTY;
        ret = dev->custom_ioctl(client, data.custom.cmd,
                                        data.custom.arg);
        break;
}
```

Customization of ION

We found:

✓ Most vendors would add their own ion heaps

✓ Most vendors would add custom ioctl commands

Vendor A

```
long msm_ion_custom_ioctl(struct ion_client *client,
                          unsigned int cmd,
                          unsigned long arg)
{
        ……
        switch (cmd) {
        case ION_IOC_CLEAN_CACHES:
        case ION_IOC_INV_CACHES:
        case ION_IOC_CLEAN_INV_CACHES:
        {
……
```

Vendor B

```
static long _ion_ioctl(struct ion_client *client,
unsigned int cmd, unsigned long arg, int
from_kernel)
{
        ……
        switch (cmd) {
        case ION_CMD_SYSTEM:
                ……
        case ION_CMD_MULTIMEDIA:
                ……
                break;
        }
……
```

Customization is proven to be vulnerable in many cases!

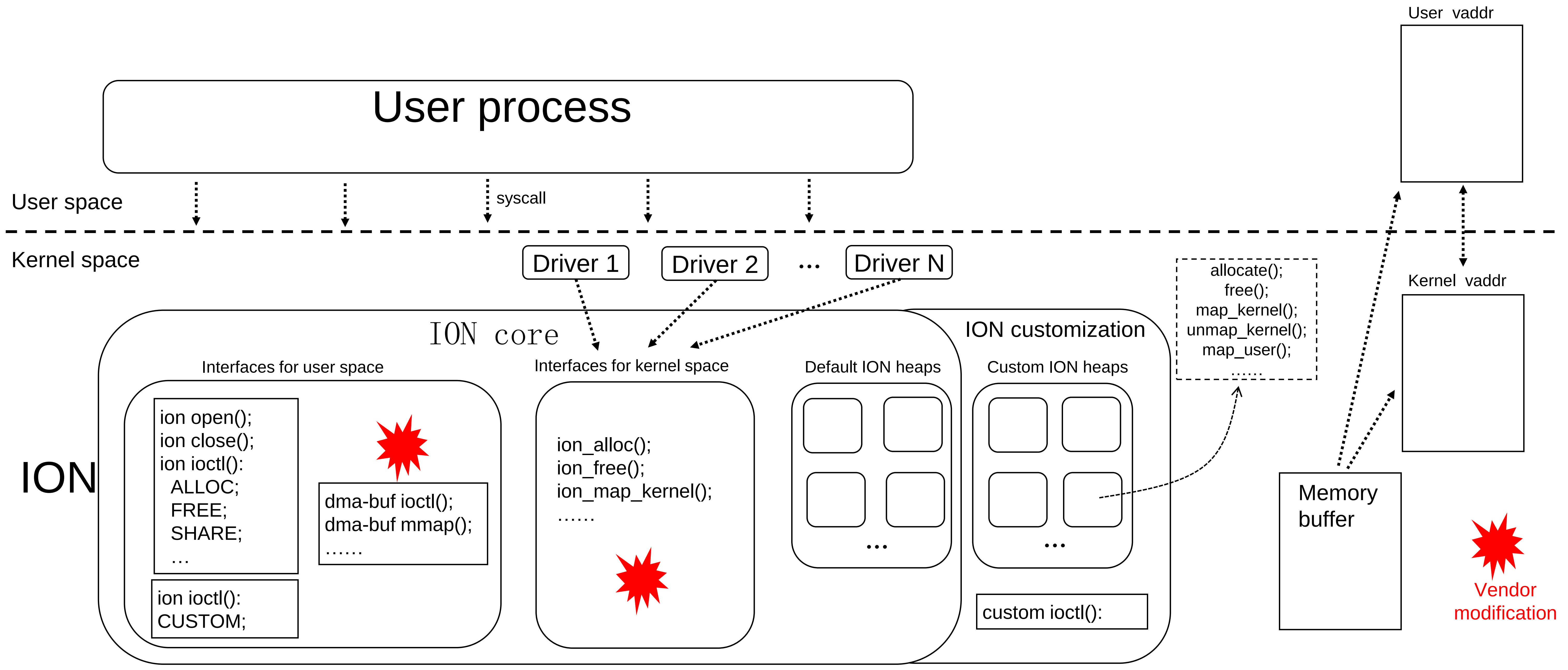
Dozens of vulnerabilities in the customization

ion_heap customization:
CVE-2021-0498,CVE-2021-0528,CVE-2021-0489,CVE-2021-0493,CVE-2021-0492,
CVE-2021-0490,CVE-2021-0496,CVE-2021-0526,CVE-2021-0420,CVE-2021-0525,
CVE-2021-0495,CVE-2021-0497,CVE-2021-0491,CVE-2021-0421,CVE-2021-0530,
CVE-2021-0494,CVE-2021-0424,CVE-2021-0527 …

ion custom ioctl:
CVE-2022-20036,CVE-2021-0419,CVE-2021-0418,CVE-2021-0529,CVE-2021-0415,
CVE-2021-0417,CVE-2021-0416,CVE-2022-20017,CVE-2022-20037,CVE-2021-0425

All the vulnerabilities have already been fixed by vendors!

Vendors’ modification to ION core



Vendors’ modification to ION core

Vendor A

Add new commands into the original ion_ioctl():

```
static long ion_ioctl(struct file *filp, unsigned int cmd,
unsigned long arg)
{
……
+               case ION_IOC_CLEAN_CACHES:
+                               return client->dev->custom_ioctl(client,
+
                ION_IOC_CLEAN_CACHES, arg);
+               case ION_IOC_INV_CACHES:
+                               return client->dev->custom_ioctl(client,
+
                ION_IOC_INV_CACHES, arg);
+               case ION_IOC_CLEAN_INV_CACHES:
+                               return client->dev->custom_ioctl(client,
+
                ION_IOC_CLEAN_INV_CACHES, arg);
……
```

Vendor B

Add some debug code into the original ion_alloc():

```
struct ion_handle *ion_alloc(struct ion_client *client, size_t len,
                             size_t align, unsigned int heap_id_mask,
                             unsigned int flags)
{
……
+ #ifdef CONFIG_MTK_ION
+               ion_history_count_kick(true, len);
+ #endif

+               handle->dbg.user_ts = end;
+               do_div(handle->dbg.user_ts, 1000000);
+               memcpy(buffer->alloc_dbg, client->dbg_name,
ION_MM_DBG_NAME_LEN);
……
```

💣 Vendors’ modification to ION core

- Vulnerable modifications ⇨ New vulnerabilities can be introduced
- Code merge conflicts could happen
  - Missing patch
  - Wrong patch

  ⇨ Known vulnerabilities are not fixed or new vulnerabilities get introduced

💣 Vendors’ modification to ION core

- Analyze the modifications made by vendors
- Investigate known issues of ION and check if they are fixed downstream

- Vulnerabilities introduced by Modification
  CVE-2021-0422 🔍
- Vulnerabilities due to the missing patch
  CVE-2021-39801 🔍
  CVE-2022-20109
- Vulnerabilities due to the wrong patch
  CVE-2022-20118 🔍
  CVE-2021-0610

CVE-2021-0422

- ➢ Affected ION version:
  A vendor’s devices with ION V1

- ➢ Root cause:
  Vulnerable modifications to ION core

- ➢ Impact:
  Illegal memory access

CVE-2021-0422

```
struct ion_handle *ion_import_dma_buf_fd(struct ion_client
*client, int fd)
{
            struct dma_buf *dmabuf;
            struct ion_handle *handle;
            ……
            dmabuf = dma_buf_get(fd);
            ……
            handle = ion_import_dma_buf(client, dmabuf);
            dma_buf_put(dmabuf);
            ……
            handle->dbg.fd = fd;
            handle->dbg.user_ts = sched_clock();
            do_div(handle->dbg.user_ts, 1000000);
            return handle;
}
```

"handle" could be an error code

"handle" is not checked, invalid memory access will happen!

CVE-2021-39801

- Affected ION version:
  ION V1

- Root cause:
  The patch of a known issue(wrong behavior of the ION API) is missing

- Impact:
  UAF

CVE-2021-39801

A known issue in ION_IOC_FREE:

```
case ION_IOC_FREE:
{
		struct ion_handle *handle;
		mutex_lock(&client->lock);
		handle = ion_handle_get_by_id_nolock(client,
							data.handle.handle);
		……
		ion_free_nolock(client, handle);
		ion_handle_put_nolock(handle);
		mutex_unlock(&client->lock);
		break;
}
```

…

```
static void ion_handle_destroy(struct kref *kref)
{
		struct ion_handle *handle = container_of(kref, struct
ion_handle, ref);
		……
		kfree(handle);
}
```

```
handle_data.handle = ion_handle_id;
while(1) {
		ioctl(client_fd, ION_IOC_FREE,
&handle_data);
}
```

ion_handle object in the kernel could be released at any time from user space!

CVE-2021-39801

UAF scenario:

```
case ION_IOC_ALLOC:
{
        struct ion_handle *handle;
        handle = ion_alloc(client, data.allocation.len,
                                        data.allocation.align,
                                        data.allocation.heap_id_mask,
                                        data.allocation.flags);
        if (IS_ERR(handle))
                return PTR_ERR(handle);
        data.allocation.handle = handle->id; UAF
        cleanup_handle = handle;
        break;
}
……
        if (copy_to_user((void __user *)arg, &data, _IOC_SIZE(cmd))) {
                if (cleanup_handle)
                        ion_free(client, cleanup_handle); UAF
                return -EFAULT;
        }
```

Are there any other places that would trigger the UAF?

CVE-2021-39801

The issue is assigned CVE-id: CVE-2017-0564 in 2017.
Patch for the issue:
ANDROID: ion: Protect kref from userspace manipulation

This separates the kref for ion handles into two components. Userspace requests through the ioctl will hold at most one reference to the internally used kref. All additional requests will increment a separate counter, and the original reference is only put once that counter hits 0. This protects the kernel from a poorly behaving userspace.

But some Android common kernel branches & Upstream kernel branches & some vendors' kernel branches missed the patch!

CVE-2022-20118

- Affected ION version:
  A vendor’s devices with ION V1
- Root cause:
  Wrong patch for a known mutex lock using issue
- Impact:
  UAF

CVE-2022-20118

A known UAF issue in the ION_IOC_SHARE :

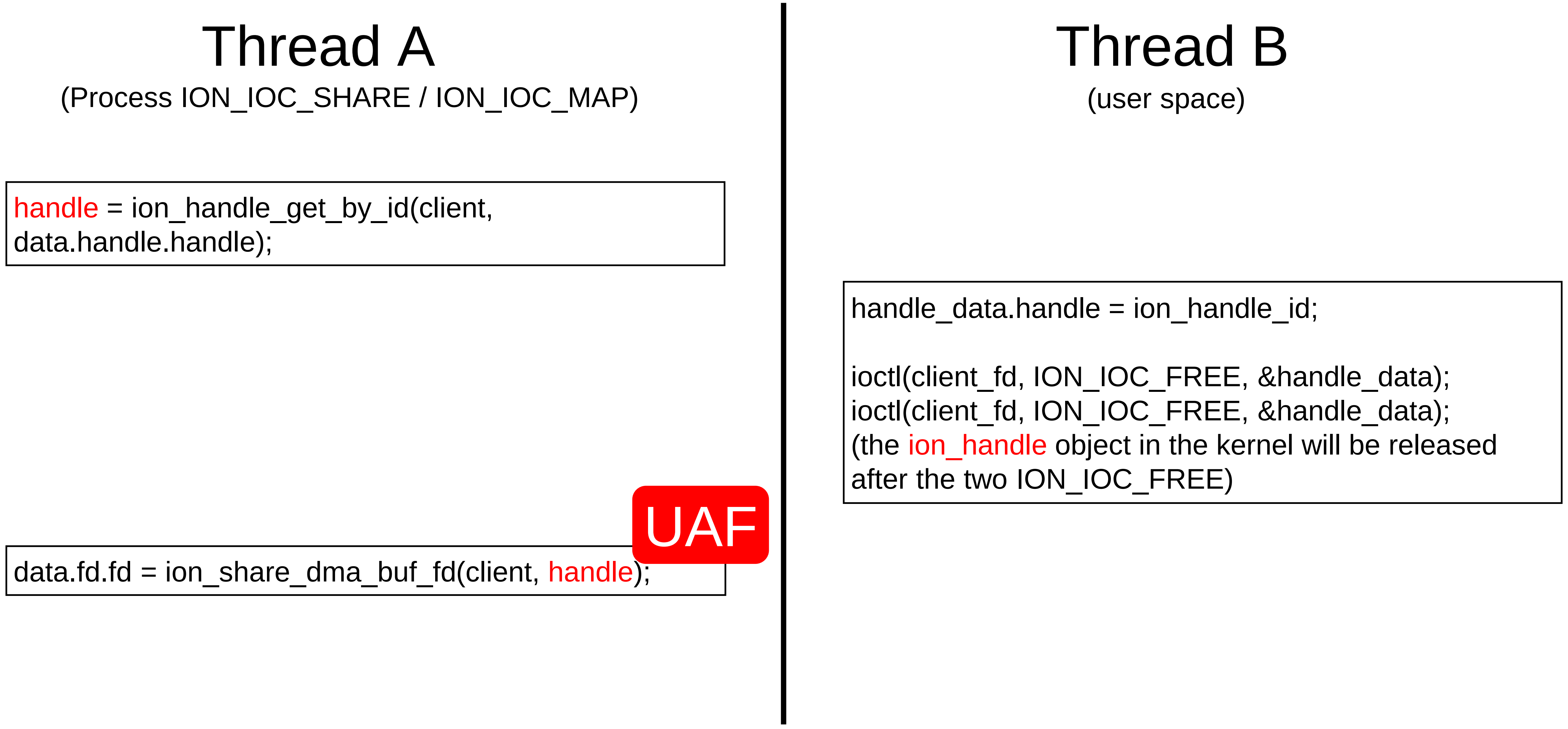
```
case ION_IOC_SHARE:
case ION_IOC_MAP:
{
		struct ion_handle *handle;
		handle = ion_handle_get_by_id(client, data.handle.handle);
		if (IS_ERR(handle))
			return PTR_ERR(handle);
		data.fd.fd = ion_share_dma_buf_fd(client, handle);
		ion_handle_put(handle);
		if (data.fd.fd < 0)
			ret = data.fd.fd;
		break;
}
```

Wrong behavior version of ION_IOC_FREE

```
case ION_IOC_FREE:
{
		struct ion_handle *handle;
		mutex_lock(&client->lock);
		handle = ion_handle_get_by_id_nolock(client,
data.handle.handle);
		if (IS_ERR(handle)) {
			mutex_unlock(&client->lock);
			return PTR_ERR(handle);
		}
		ion_free_nolock(client, handle);
		ion_handle_put_nolock(handle);
		mutex_unlock(&client->lock);
		break;
}
```

CVE-2022-20118

A known UAF issue in the ION_IOC_SHARE:



Thread A

(Process ION_IOC_SHARE / ION_IOC_MAP)

```
handle = ion_handle_get_by_id(client, data.handle.handle);
```

Thread B

(user space)

```
handle_data.handle = ion_handle_id;

ioctl(client_fd, ION_IOC_FREE, &handle_data);
ioctl(client_fd, ION_IOC_FREE, &handle_data);
(the ion_handle object in the kernel will be released after the two ION_IOC_FREE)
```

Thread A

```
data.fd.fd = ion_share_dma_buf_fd(client, handle);
```

CVE-2022-20118

The patch to fix the issue in 2018:

staging: android: ion: fix ION_IOC_{MAP,SHARE} use-after-free:

```
case ION_IOC_SHARE:
case ION_IOC_MAP:
{
        struct ion_handle *handle;
        mutex_lock(&client->lock);
        handle = ion_handle_get_by_id_nolock(client, data.handle.handle);
        if (IS_ERR(handle)) {
                mutex_unlock(&client->lock);
                return PTR_ERR(handle);
        }
        data.fd.fd = ion_share_dma_buf_fd_nolock(client, handle);
        ion_handle_put_nolock(handle);
        mutex_unlock(&client->lock);
        if (data.fd.fd < 0)
                ret = data.fd.fd;
        break;
}
```

```
case ION_IOC_FREE:
{
        struct ion_handle *handle;
        mutex_lock(&client->lock);
        handle = ion_handle_get_by_id_nolock(client,
data.handle.handle);
        if (IS_ERR(handle)) {
                mutex_unlock(&client->lock);
                return PTR_ERR(handle);
        }
        ion_free_nolock(client, handle);
        ion_handle_put_nolock(handle);
        mutex_unlock(&client->lock);
        break;
}
```

CVE-2022-20118

A variant vulnerability similar to the known issue:

In a vendor’s ION deriver:

```
case ION_IOC_SHARE:
case ION_IOC_MAP:
{
        struct ion_handle *handle;

        handle = ion_handle_get_by_id_nolock(client,
data.handle.handle);
        if (IS_ERR(handle))
                return PTR_ERR(handle);
        data.fd.fd = ion_share_dma_buf_fd(client, handle);
        ion_handle_put(handle);
        if (data.fd.fd < 0)
                ret = data.fd.fd;
        break;
}
```

```
struct ion_handle *ion_handle_get_by_id_nolock(struct ion_client
*client,
                                                      int id)
{
        struct ion_handle *handle;
        handle = idr_find(&client->idr, id);
        if (handle)
                ion_handle_get(handle);
        return handle ? handle : ERR_PTR(-EINVAL);
}
```

```
case ION_IOC_FREE:
(correct behavior version)
```

CVE-2022-20118

UAF still happens in function ion_handle_get_by_id_nolock:

Thread A

(Enter function ion_handle_get_by_id_nolock)

```
handle = idr_find(&client->idr, id);
```

```
if (handle)
                         ion_handle_get(handle)
```

UAF

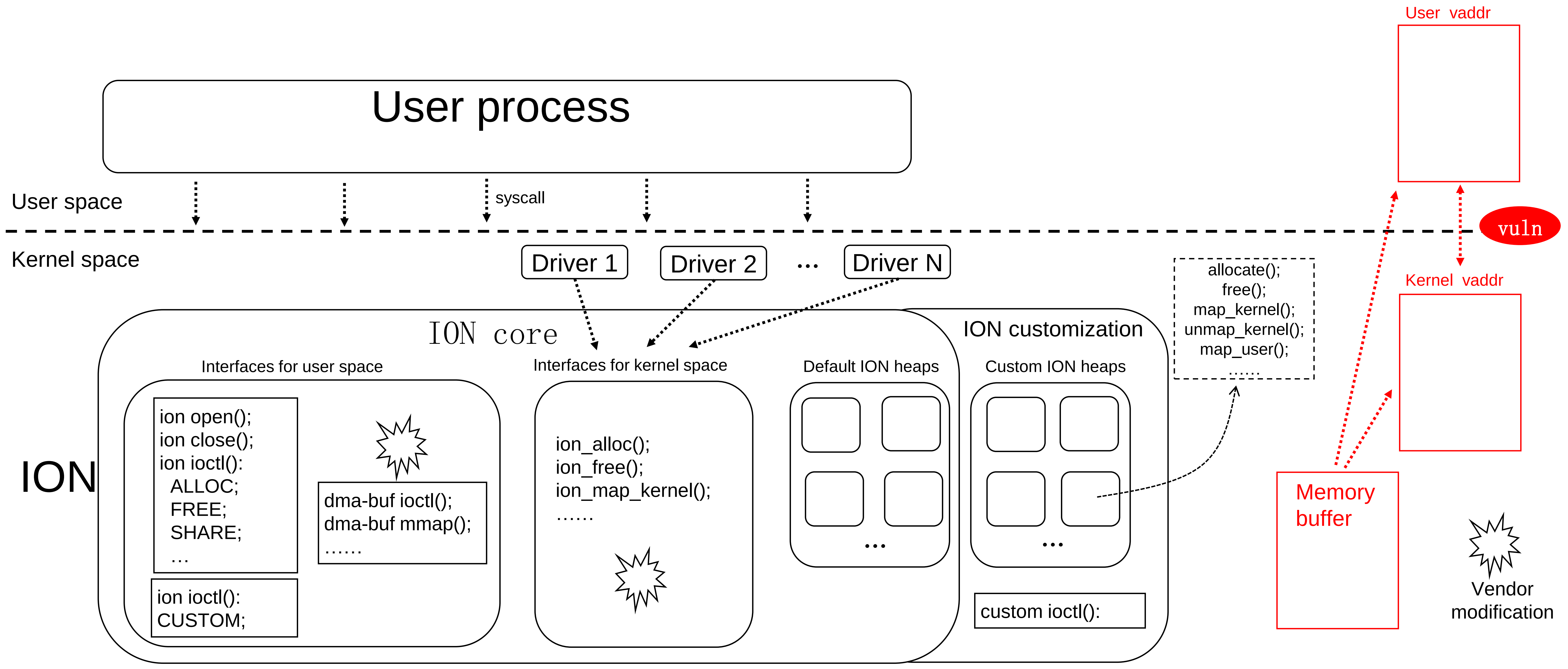
Thread B

(User space)

```
handle_data.handle = ion_handle_id;

ioctl(client_fd, ION_IOC_FREE, &handle_data); (the
ion_handle object in the kernel will be released after the
ION_IOC_FREE)
```

Buffer sharing



Buffer sharing feature of ION can introduce vulnerabilities:

- 《Android ION Hazard》

Two kinds of vulnerabilities introduced by ION buffering sharing

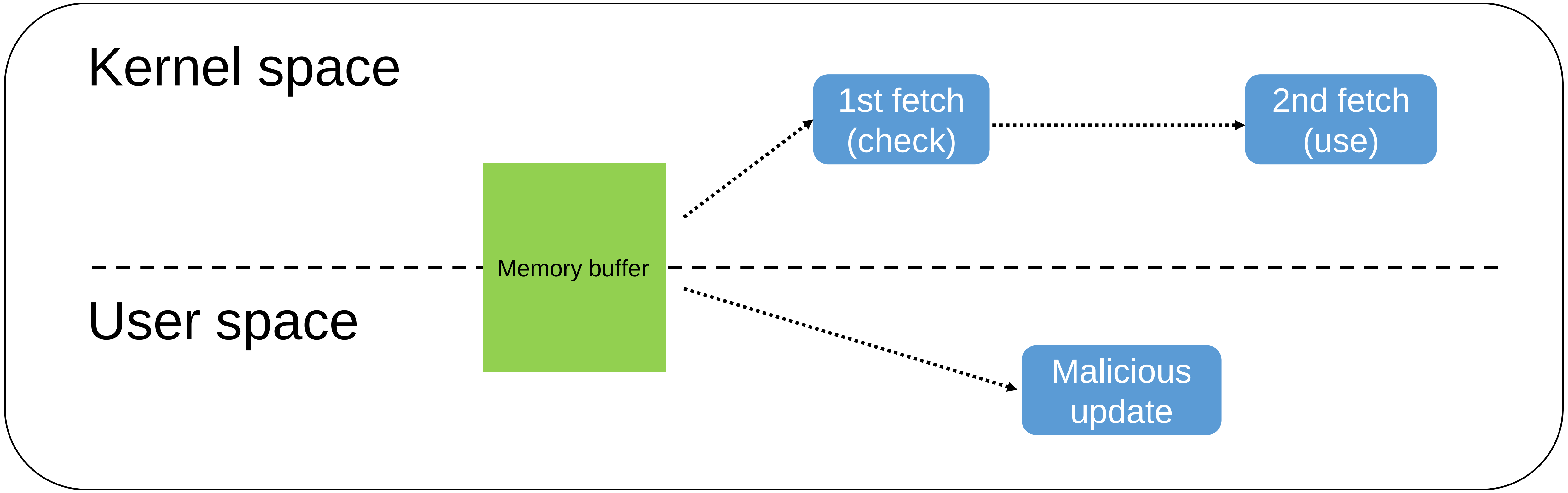
- System crash due to hardware protection

- Sensitive information leakage

Buffer sharing feature of ION can introduce vulnerabilities:

➢ 《An iOS hacker tries Android》

- Double fetch vulnerabilities introduced by Buffer Sharing



Double fetch vulnerabilities introduced by Buffer Sharing

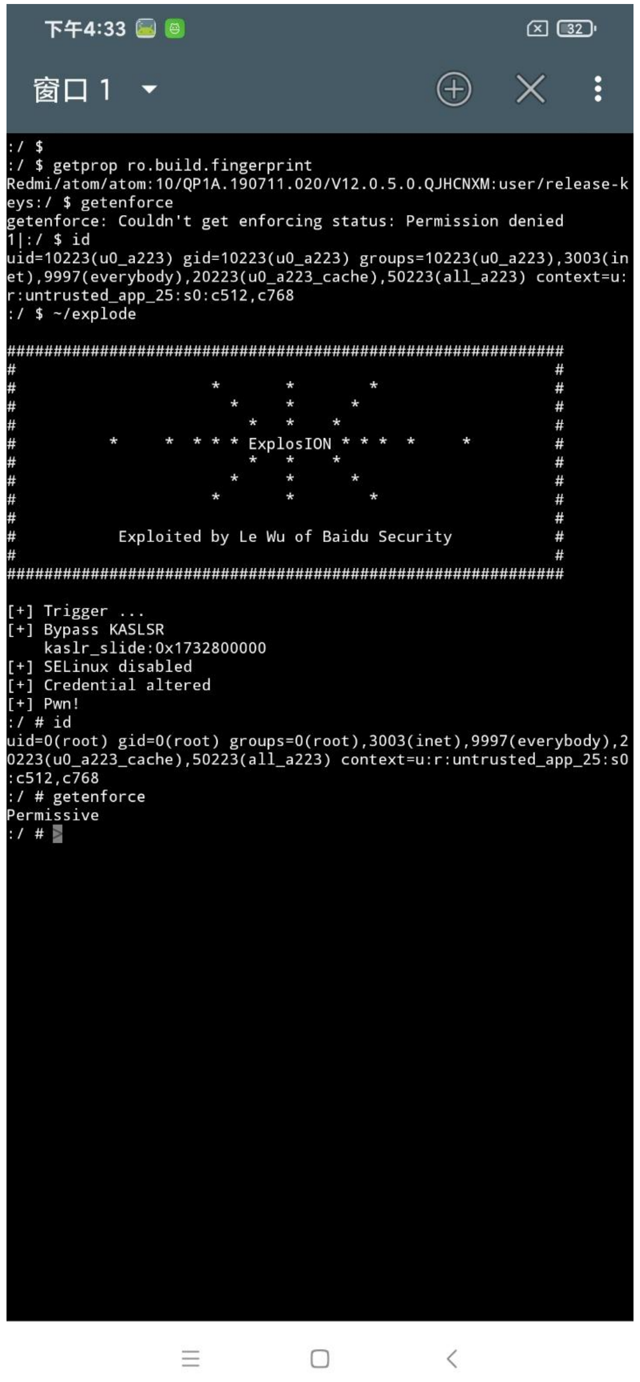
We found other double fetch vulnerabilities in a vendor’s apu driver：

CVE-2021-0897
CVE-2021-0895
CVE-2021-0903

Is ION the only component that has buffer sharing feature?

Exploitation

Get root!



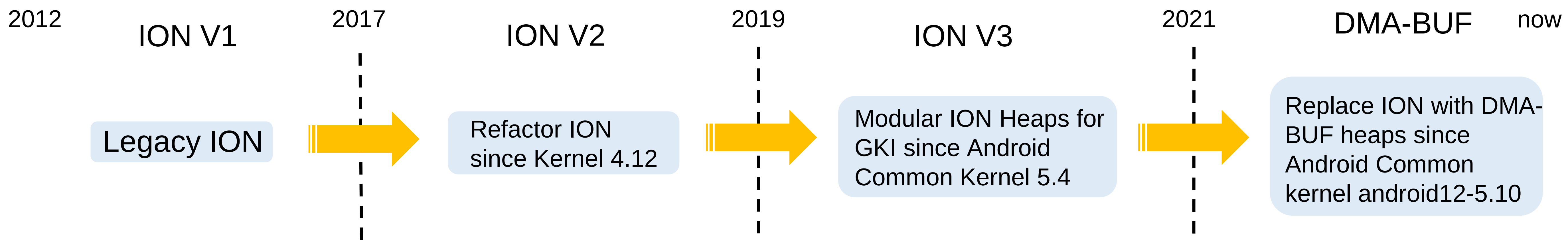
```
:/ $
:/ $ getprop ro.build.fingerprint
Redmi/atom/atom:10/QP1A.190711.020/V12.0.5.0.QJHCNXM:user/release-k
eys:/ $ getenforce
getenforce: Couldn't get enforcing status: Permission denied
1|:/ $ id
uid=10223(u0_a223) gid=10223(u0_a223) groups=10223(u0_a223),3003(in
et),9997(everybody),20223(u0_a223_cache),50223(all_a223) context=u:
r:untrusted_app_25:s0:c512,c768
:/ $ ~/explode

############################################################
#                                                          #
#                     *       *        *                   #
#                       *     *      *                     #
#                         *   *    *                       #
#          *     *  * * * ExplosION * * *  *     *         #
#                         *   *    *                       #
#                       *     *      *                     #
#                     *       *        *                   #
#                                                          #
#           Exploited by Le Wu of Baidu Security           #
#                                                          #
############################################################

[+] Trigger ...
[+] Bypass KASLSR
    kaslr_slide:0x1732800000
[+] SELinux disabled
[+] Credential altered
[+] Pwn!
:/ # id
uid=0(root) gid=0(root) groups=0(root),3003(inet),9997(everybody),2
0223(u0_a223_cache),50223(all_a223) context=u:r:untrusted_app_25:s0
:c512,c768
:/ # getenforce
Permissive
:/ #
```

Reflections on ExplosION

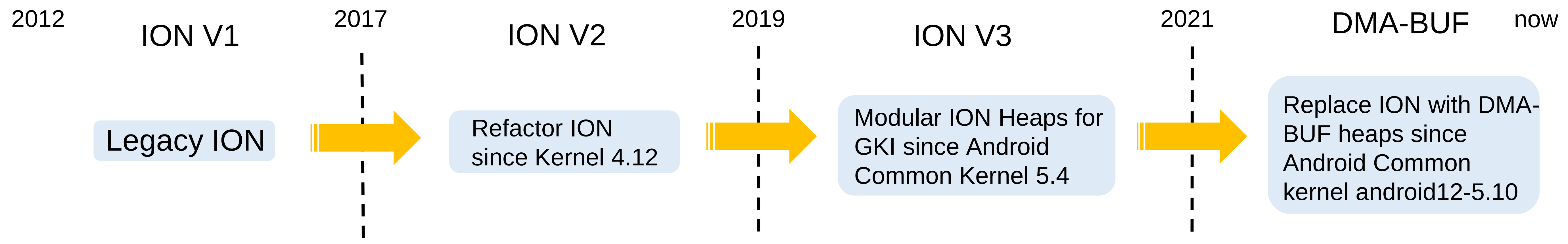
➢ ION Evolution——Refactors from Upstream


2012
ION V1
Legacy ION
2017
ION V2
Refactor ION since Kernel 4.12
2019
ION V3
Modular ION Heaps for GKI since Android Common Kernel 5.4
2021
DMA-BUF
now
Replace ION with DMA-BUF heaps since Android Common kernel android12-5.10

- Have been used since around 2012
- Still widely used in devices of which kernel<4.12
- Bug fixes & small refactors happen from time to time

Reflections on ExplosION

- ION Evolution——Refactors from Upstream



Legacy ION

Refactor ION since Kernel 4.12

Modular ION Heaps for GKI since Android Common Kernel 5.4

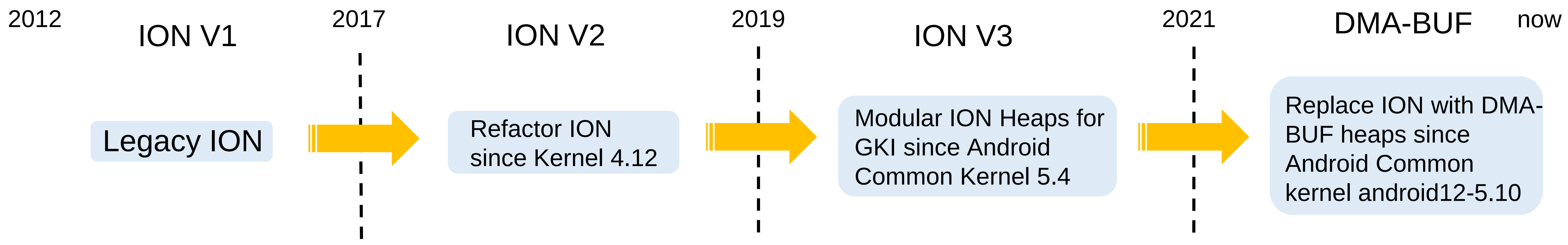
Replace ION with DMA-BUF heaps since Android Common kernel android12-5.10

- Removal of Ion clients and handles: IOC_ION_ALLOC ioctl directly outputs dma-buf fds.
- Addition of cache-coherency ioctls: Kernel 4.12 replaced ION_IOC_SYNC with the DMA_BUF_IOCTL_SYNC ioctl

ION core become more brief,which means more secure!

Reflections on ExplosION

- ION Evolution——Refactors from Upstream

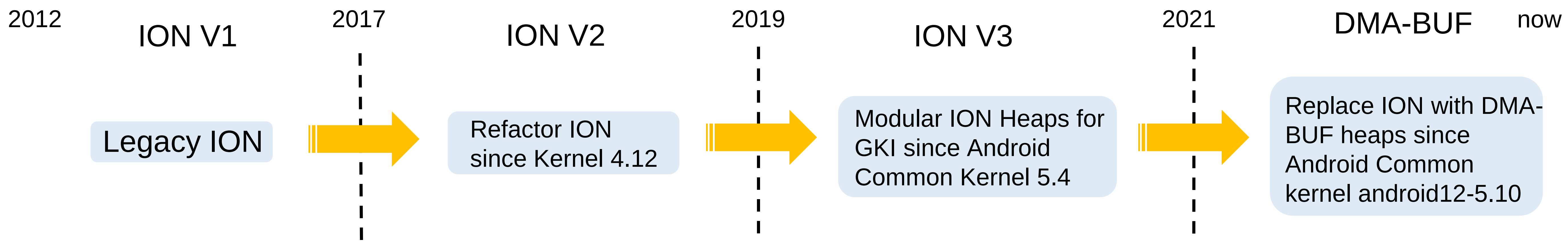

2012
ION V1
2017
ION V2
2019
ION V3
2021
DMA-BUF
now
Legacy ION
Refactor ION since Kernel 4.12
Modular ION Heaps for GKI since Android Common Kernel 5.4
Replace ION with DMA-BUF heaps since Android Common kernel android12-5.10

No more vendor modifications!

- The ION core driver can be part of the GKI image, enabling all device-independent performance optimizations and bug fixes to reach all devices.
- The ION core driver in the common kernel can handle heap registration and manage the interface to userspace and kernel clients. The vendor heap modules are required only to implement the custom heap operations.

Reflections on ExplosION

➢ ION Evolution——Refactors from Upstream



2012 ION V1

Legacy ION

2017 ION V2

Refactor ION since Kernel 4.12

2019 ION V3

Modular ION Heaps for GKI since Android Common Kernel 5.4

2021 DMA-BUF now

Replace ION with DMA-BUF heaps since Android Common kernel android12-5.10

➢ Replace the ION with DMA-BUF

Advantages of DMA-BUF heaps:

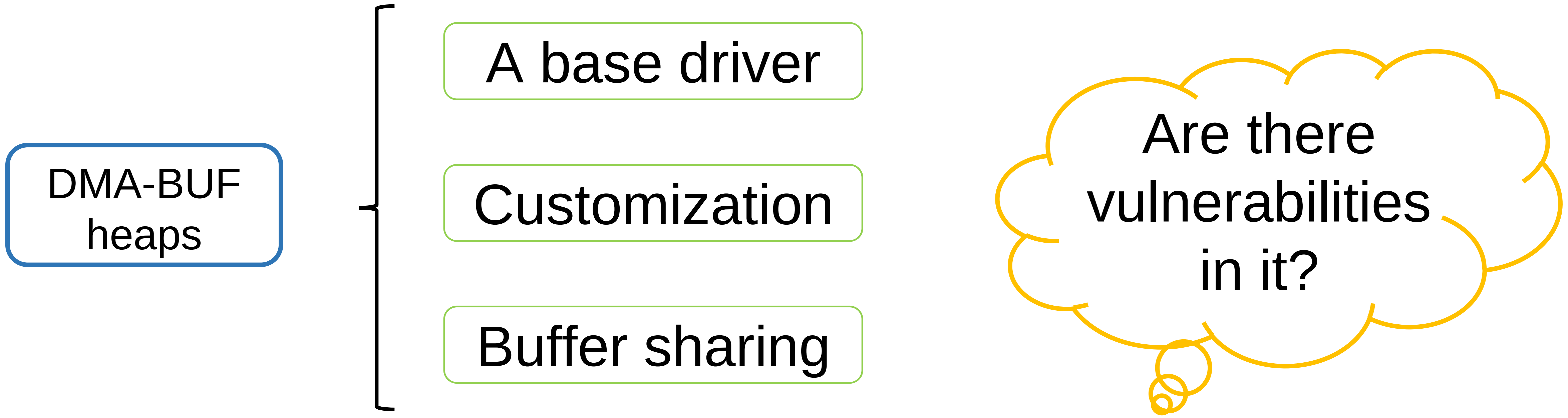
✓ Security
✓ ABI stability
✓ Standardization

Reflections on ExplosION

- Suggestions for vendors
  - Apply the refactors right now!
  - Patch quickly and carefully!
  - Understand the ION APIs before using them!
  - Do customization carefully!

All the affected vendors had worked diligently with us to remediate the ExplosION and had already made the patches available.

Future work


DMA-BUF heaps
A base driver
Customization
Buffer sharing
Are there vulnerabilities in it?

Acknowledge

- Thanks to Chengfu Bao, 某因幡, Shufan Yang, Lin Wu

Thank you!

Supplement

Details of other ION vulnerabilities

CVE-2021-39714

- ➢ Affected ION version:
  ION V1,V2

- ➢ Root cause:
  Reference count overflow in ION core

- ➢ Impact:
  Potential UAF would be introduced into vendor drivers by ION

CVE-2021-39714

```
void *ion_map_kernel(struct ion_client *client, struct
ion_handle *handle)
{
        struct ion_buffer *buffer;
        void *vaddr;
        mutex_lock(&client->lock);
        ……
        buffer = handle->buffer;
        if (!handle->buffer->heap->ops->map_kernel) {
                ……
        }
        mutex_lock(&buffer->lock);
        vaddr = ion_handle_kmap_get(handle);
        mutex_unlock(&buffer->lock);
        mutex_unlock(&client->lock);
        return vaddr;
}
```

……

```
static void *ion_buffer_kmap_get(struct ion_buffer *buffer)
{
        void *vaddr;
        if (buffer->kmap_cnt) {
                buffer->kmap_cnt++;
                return buffer->vaddr;
        }
        vaddr = buffer->heap->ops->map_kernel(buffer-
>heap, buffer);
        ……
        buffer->vaddr = vaddr;
        buffer->kmap_cnt++;
        return vaddr;
}
```

Integer overflow can happen !!!

CVE-2021-39714

```
void ion_unmap_kernel(struct ion_client *client, struct
ion_handle *handle)
{
        struct ion_buffer *buffer;
        mutex_lock(&client->lock);
        buffer = handle->buffer;
        mutex_lock(&buffer->lock);
        ion_handle_kmap_put(handle);
        mutex_unlock(&buffer->lock);
        mutex_unlock(&client->lock);
}
```

……

```
static void ion_buffer_kmap_put(struct ion_buffer *buffer)
{
        buffer->kmap_cnt--;
        if (!buffer->kmap_cnt) {
                buffer->heap->ops->unmap_kernel(buffer->heap,
buffer);
                buffer->vaddr = NULL;
        }
}
```

CVE-2021-39714

Thread A	Thread B
call ion_map_kernel() constantly to let ion_buffer->kmap_cnt become 0xffffffff	
	vaddr = ion_map_kernel(); ion_buffer->kmap_cnt become 0; Integer overflow
vaddr2 = ion_map_kernel(); ion_buffer->kmap_cnt become 1;	
	ion_unmap_kernel(); vaddr2 will be unmapped
Access the vaddr2; UAF	

CVE-2022-20109

- Affected ION version:
  ION V1

- Root cause:
  The patch of a known issue is missing, resulting in a reference count issue of ion_handle

- Impact:
  UAF

CVE-2022-20109

The latest legacy ION_IOC_ALLOC:

```
case ION_IOC_ALLOC:
{
        struct ion_handle *handle;
        handle = __ion_alloc(client, data.allocation.len,
                        data.allocation.align,
                        data.allocation.heap_id_mask,
                        data.allocation.flags, true);
        ……
        cleanup_handle = handle;
        pass_to_user(handle);
        break;
}
……
        if (copy_to_user((void __user *)arg, &data, _IOC_SIZE(cmd))) {
                if (cleanup_handle) {
                        mutex_lock(&client->lock);
                        user_ion_free_nolock(client, cleanup_handle);
                        ion_handle_put_nolock(cleanup_handle);
                        mutex_unlock(&client->lock);
……
```

Reference count of Ion_handle becomes 2

Reference count of ion_handle becomes 1

Reference count of Ion_handle becomes 0,ion_handle will be released!

CVE-2022-20109

ION_IOC_ALLOC of a vendor’s devices is like this:

```
case ION_IOC_ALLOC:
{
        struct ion_handle *handle;

        handle = ion_alloc(client, data.allocation.len,
                           data.allocation.align,
                           data.allocation.heap_id_mask,
                           data.allocation.flags);
        ……
        pass_to_user(handle);
        data.allocation.handle = handle->id;
        cleanup_handle = handle;
……
if (copy_to_user((void __user *)arg, &data, _IOC_SIZE(cmd))) {
                if (cleanup_handle) {
                        mutex_lock(&client->lock);
                        user_ion_free_nolock(client, cleanup_handle);
                        ion_handle_put_nolock(cleanup_handle);
……
```

Reference count of Ion_handle becomes 1

Reference count of Ion_handle becomes 0,ion_handle will be released!

UAF

CVE-2022-20109

The root cause of the UAF: Missed a patch which is released in **2016**

ion: Fix use after free during ION_IOC_ALLOC

If a user happens to call ION_IOC_FREE during an ION_IOC_ALLOC
on the just allocated id, and the copy_to_user fails, the cleanup
code will attempt to free an already freed handle.

This adds a wrapper for ion_alloc that adds an ion_handle_get to
avoid this.

CVE-2021-0610

- Affected ION version:
  ION V1

- Root cause:
  The patch of a known issue(ion_handle kref overflow) is missing

- Impact:

CVE-2021-0610

A known issue 3 years ago:

```
case ION_IOC_IMPORT:
{
        struct ion_handle *handle;
        handle = ion_import_dma_buf_fd(client, data.fd.fd);
        if (IS_ERR(handle)) {
                ret = PTR_ERR(handle);
        } else {
                data.handle.handle = handle->id;
                handle = pass_to_user(handle);
                if (IS_ERR(handle)) {
                        ret = PTR_ERR(handle);
                        data.handle.handle = 0;
                }
        }
        break;
}
```

```
struct ion_handle *ion_import_dma_buf(struct ion_client *client,
                                      struct dma_buf *dmabuf)
{
        struct ion_buffer *buffer;
        struct ion_handle *handle;
        int ret;
        ……
        buffer = dmabuf->priv;
        mutex_lock(&client->lock);
        /* if a handle exists for this buffer just take a reference to it */
        handle = ion_handle_lookup(client, buffer);
        if (!IS_ERR(handle)) {
                ion_handle_get(handle);
                mutex_unlock(&client->loc   kref overflow !
                goto end;
        }
        ……
}
```

CVE-2021-0610

The patch to fix it:

staging: android: ion: check for kref overflow

This patch is against 4.9. It does not apply to master due to a large
rework of ion in 4.12 which removed the affected functions altogther.
4c23cbff073f3b9b ("staging: android: ion: Remove import interface")

Userspace can cause the kref to handles to increment arbitrarily high. Ensure it does not overflow.

```
+/* Must hold the client lock */
+static struct ion_handle *ion_handle_get_check_overflow(
+                                        struct ion_handle *handle)
+{
+        if (atomic_read(&handle->ref.refcount) + 1 == 0)
+                return ERR_PTR(-EOVERFLOW);
+        ion_handle_get(handle);
+        return handle;
+}
+
……
 static bool ion_handle_validate(struct ion_client *client,
@@ -1110,7 +1121,7 @@
        /* if a handle exists for this buffer just take a reference to it */
        handle = ion_handle_lookup(client, buffer);
        if (!IS_ERR(handle)) {
-               ion_handle_get(handle);
+               handle = ion_handle_get_check_overflow(handle);
                mutex_unlock(&client->lock);
                goto end;
        }
```

CVE-2021-0610

The issue still exists in a vendor's devices because of the wrong patch:

```
struct ion_handle *ion_import_dma_buf(struct ion_client *client,
                                      struct dma_buf *dmabuf)
{
        struct ion_buffer *buffer;
        struct ion_handle *handle;
        ……
        buffer = dmabuf->priv;

        mutex_lock(&client->lock);
        /* if a handle exists for this buffer just take a reference to it */
        handle = ion_handle_lookup(client, buffer);
        if (!IS_ERR(handle)) {
                ion_handle_get_check_overflow(handle);
                mutex_unlock(&client->lock);
                goto end;
        }
        ……
}
```

```
static struct ion_handle *ion_handle_get_check_overflow(

        struct ion_handle *handle)
{
        if (atomic_read(&handle->ref.refcount.refs) + 1 ==
0)
                return ERR_PTR(-EOVERFLOW);
        ion_handle_get(handle);
        return handle;
}
```

Kref overflow can still happen!!!