Calif Research

# Privilege Escalation via a Page Use-After-Free in Qualcomm's QAIC Driver

SPEAKER
Lukas Maar

CONFERENCE
PRASEC

DATE
September 10, 2026

# Who Am I

## Lukas Maar

**Security Researcher — Calif.io & ISEC, TU Graz**

- Zero-days in the Linux & Android kernels, Android apps
- KernelSnitch — bypassing heap KASLR & MTE (Pwnie Award 2025 nominee)
- USENIX Security, NDSS, Black Hat USA/Asia, Nullcon

lukas.maar@calif.io

lukasmaar.github.io/slides/prasec26-qaic-page-uaf.pdf

# One Bug, Three Parts

Bug in Qualcomm's AI Driver

Getting Root

# One Bug, Three Parts

Bug in Qualcomm's AI Driver

01

**Driver Virtualization**

fake QAIC device in QEMU

Getting Root

# One Bug, Three Parts

**Bug in Qualcomm's AI Driver**

**01**
**Driver Virtualization**
`fake QAIC device in QEMU`

**02**
**Trigger Bug**
`missing VMA bound check`

**Getting Root**

# One Bug, Three Parts

**Bug in Qualcomm's AI Driver**

**01**
**Driver Virtualization**
`fake QAIC device in QEMU`

**02**
**Trigger Bug**
`missing VMA bound check`

**03**
**Exploitation**
`UAF → arb r/w → root`

**Getting Root**

# Background

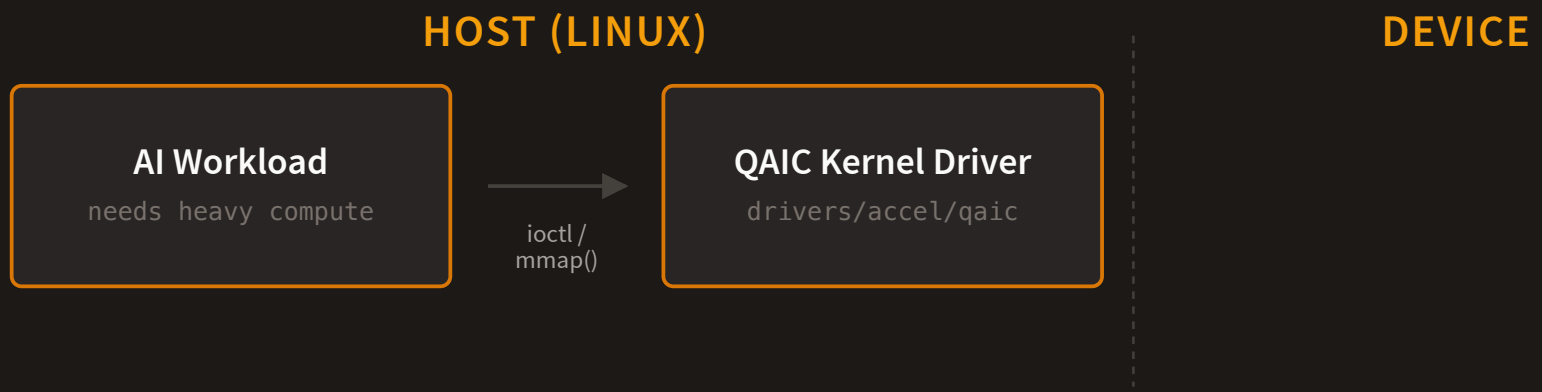
# What Is Qualcomm's Cloud AI Accelerator (QAIC)

HOST (LINUX)

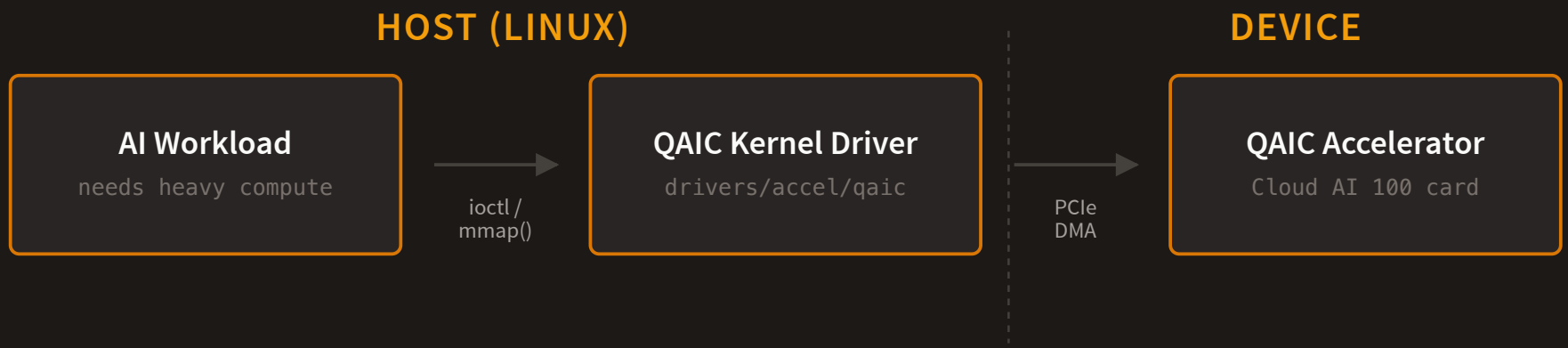
DEVICE

# What Is Qualcomm's Cloud AI Accelerator (QAIC)

# What Is Qualcomm's Cloud AI Accelerator (QAIC)



# What Is Qualcomm's Cloud AI Accelerator (QAIC)



# What Is Qualcomm's Cloud AI Accelerator (QAIC)

# Driver Virtualization

# How to

AI-assisted fake-online patch generation

# How to

AI-assisted fake-online patch generation

# What is changed

```
int __init qaic_init(void)
{
    [...]
    return 0;
}

void __exit qaic_deinit(void)
{
    [...]
}
```

# What is changed

```
int __init qaic_init(void)
{
    [...]
    return 0;
}

void __exit qaic_deinit(void)
{
    [...]
}
```

qaic_init / qaic_deinit: driver's constructor and destructor

# What is changed

```
+ static bool qaic_fake_online;   // true once the fake device is live
+ // allocate everything except stuff required by hardware
+ struct qaic_device *create_fake_qdev(void)
+ {
+     struct qaic_device *qdev = qaic_device_alloc();
+     qdev->dev_state  = QAIC_ONLINE;
+     return qdev;
+ }
+ static void destroy_fake_qdev(void) { [...] }

  int __init qaic_init(void)
  {
+     if (qaic_fake_online)
+         create_fake_qdev();
      [...]
      return 0;
  }

  void __exit qaic_deinit(void)
  {
+     if (qaic_fake_online)
+         destroy_fake_qdev();
      [...]
  }
```

# What is changed

```
static void qaic_postclose(struct drm_device *dev, struct drm_file *file)
{
    [...]
    qdev = qddev->qdev;
    qdev_rcu_id = srcu_read_lock(&qdev->dev_lock);
    if (qdev->dev_state == QAIC_ONLINE) {
        qaic_release_usr(qdev, usr);
        for (i = 0; i < qdev->num_dbc; ++i)
            if (qdev->dbc[i].usr && qdev->dbc[i].usr->handle == usr->handle)
                release_dbc(qdev, i);
    }
    srcu_read_unlock(&qdev->dev_lock, qdev_rcu_id);
    [...]
}
```

# What is changed

```
static void qaic_postclose(struct drm_device *dev, struct drm_file *file)
{
    [...]
    qdev = qddev->qdev;
    qdev_rcu_id = srcu_read_lock(&qdev->dev_lock);
    if (qdev->dev_state == QAIC_ONLINE) {
        qaic_release_usr(qdev, usr);
        for (i = 0; i < qdev->num_dbc; ++i)
            if (qdev->dbc[i].usr && qdev->dbc[i].usr->handle == usr->handle)
                release_dbc(qdev, i);
    }
    srcu_read_unlock(&qdev->dev_lock, qdev_rcu_id);
    [...]
}
```

qaic_release_usr: talks to the physical device

# What is changed

```
 static void qaic_postclose(struct drm_device *dev, struct drm_file *file)
 {
     [...]
     qdev = qddev->qdev;
     qdev_rcu_id = srcu_read_lock(&qdev->dev_lock);
     if (qdev->dev_state == QAIC_ONLINE) {
-        qaic_release_usr(qdev, usr);
-        for (i = 0; i < qdev->num_dbc; ++i)
-            if (qdev->dbc[i].usr && qdev->dbc[i].usr->handle == usr->handle)
-                release_dbc(qdev, i);
+        if (!qaic_fake_online) {
+            qaic_release_usr(qdev, usr);
+            for (i = 0; i < qdev->num_dbc; ++i)
+                if (qdev->dbc[i].usr && qdev->dbc[i].usr->handle == usr->handle)
+                    release_dbc(qdev, i);
+        }
     }
     srcu_read_unlock(&qdev->dev_lock, qdev_rcu_id);
     [...]
 }
```

# Trigger Bug

# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{









}
```

# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{


}
```

DRM: kernel framework for GPU-like devices

# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{



}
```

DRM: kernel framework for GPU-like devices

GEM: DRM's memory manager for buffer objects

# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{



}
```

DRM: kernel framework for GPU-like devices

GEM: DRM's memory manager for buffer objects

vm_area_struct: kernel's descriptor for this mapping

# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;









}
```

# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;

        bo: QAIC's own buffer object

}
```

# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;









}
```

# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    sg: one chunk of the buffer's memory

}
```

# qaic_gem_object_mmap()

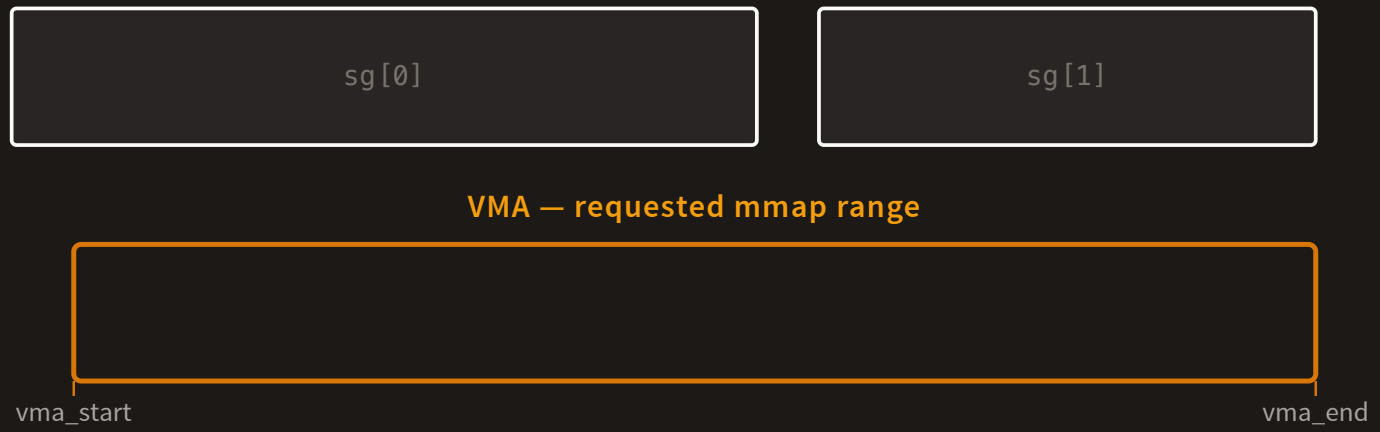
```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {









    }

    [...]
}
```

# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {
        int ret = remap_pfn_range(vma,
                                  vma->vm_start + offset,
                                  page_to_pfn(sg_page(sg)),
                                  sg->length,
                                  vma->vm_page_prot);
        if (ret)
            return ret;
        offset += sg->length;
    }

    [...]
}
```

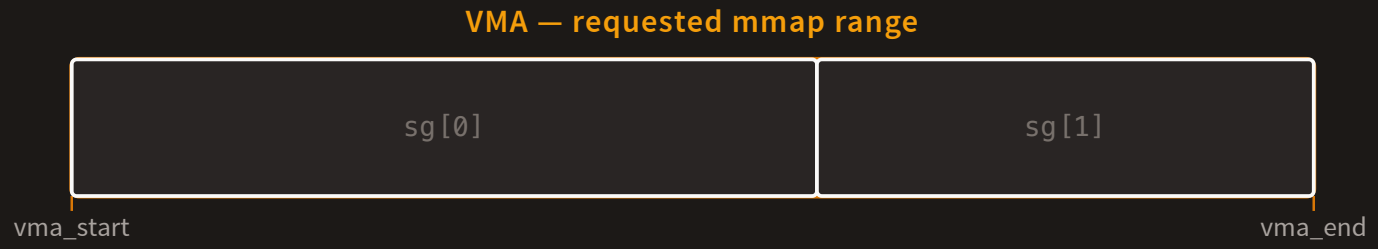
# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {
        int ret = remap_pfn_range(vma,
                                  vma->vm_start + offset,
                                  page_to_pfn(sg_page(sg)),
                                  sg->length,
                                  vma->vm_page_prot);
        if (ret)
            return ret;
        offset += sg->length;
    }

    [...]
}
```



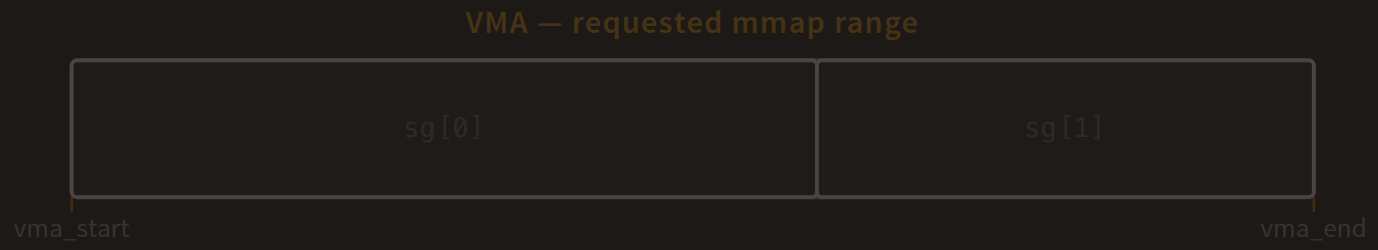
# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {
        int ret = remap_pfn_range(vma,
                                  vma->vm_start + offset,
                                  page_to_pfn(sg_page(sg)),
                                  sg->length,
                                  vma->vm_page_prot);
        if (ret)
            return ret;
        offset += sg->length;
    }

    [...]
}
```



# qaic_gem_object_mmap()

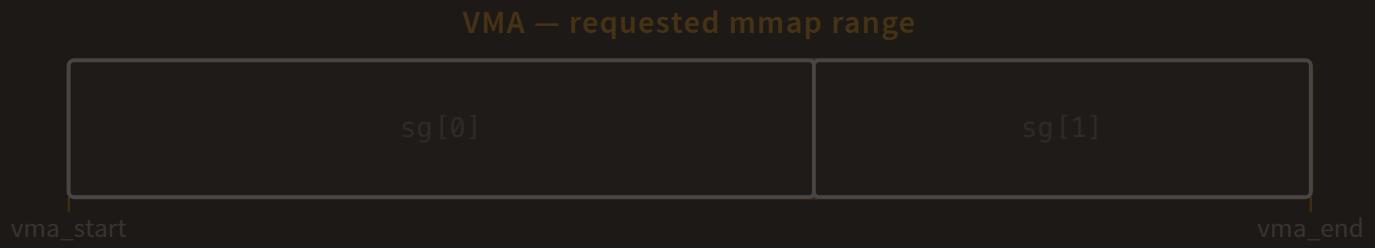
```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {
        int ret = remap_pfn_range(vma,
                                  vma->vm_start + offset,
                                  page_to_pfn(sg_page(sg)),
                                  sg->length,
                                  vma->vm_page_prot);
        if (ret)
            return ret;
        offset += sg->length;
    }

    [...]
}
```



# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {
        int ret = remap_pfn_range(vma,
                                  vma->vm_start + offset,
                                  page_to_pfn(sg_page(sg)),
                                  sg->length,
                                  vma->vm_page_prot);
        if (ret)
            return ret;
        offset += sg->length;
    }

    [...]
}
```

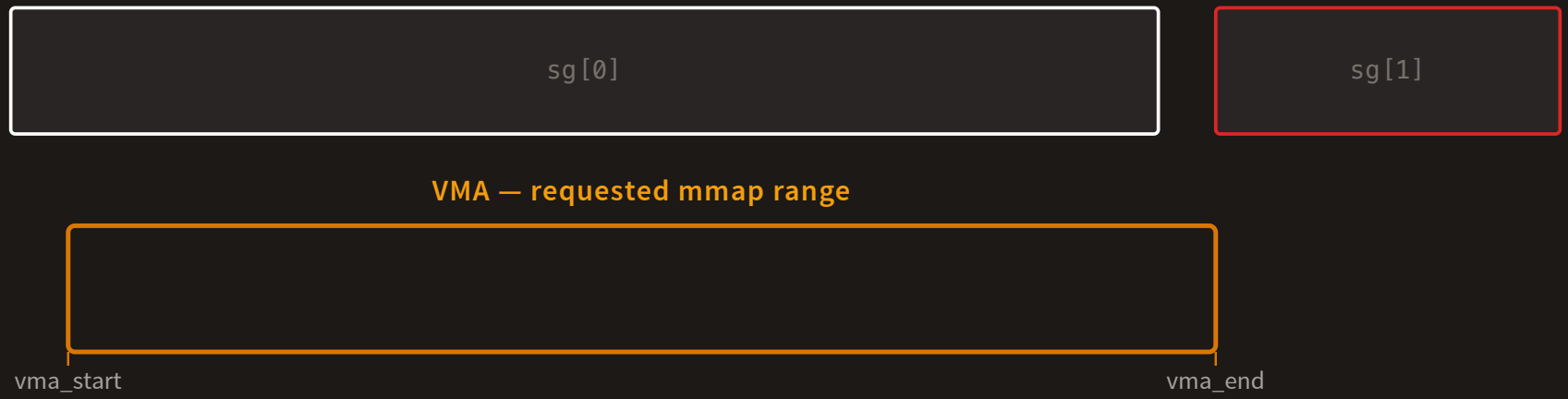
missing bounds check against vma_end



# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {
        int ret = remap_pfn_range(vma,
                                  vma->vm_start + offset,
                                  page_to_pfn(sg_page(sg)),
                                  sg->length,
                                  vma->vm_page_prot);
        if (ret)
            return ret;
        offset += sg->length;
    }

    [...]
}
```

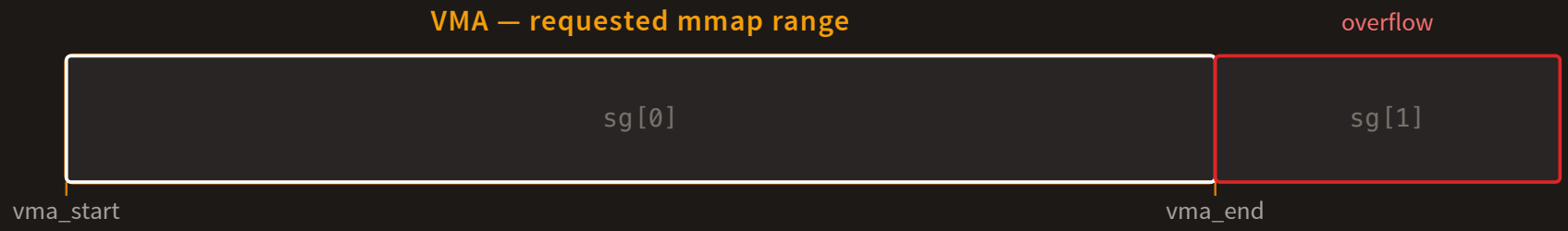
missing bounds check against vma_end



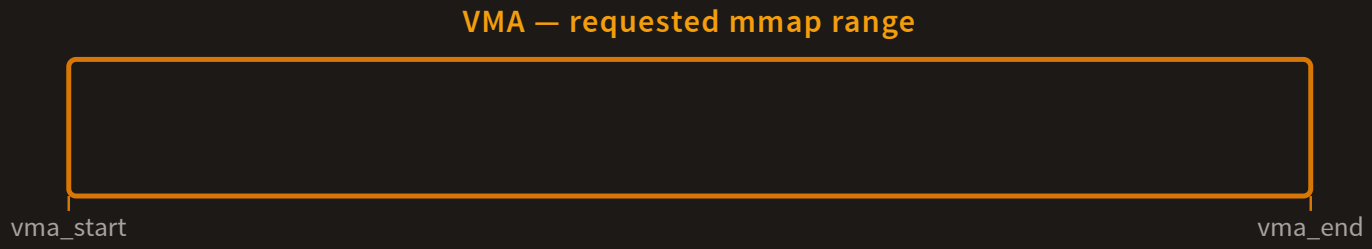
# qaic_gem_object_mmap()

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {
        int ret = remap_pfn_range(vma,
                                  vma->vm_start + offset,
                                  page_to_pfn(sg_page(sg)),
                                  sg->length,
                                  vma->vm_page_prot);
        if (ret)
            return ret;
        offset += sg->length;
    }

    [...]
}
```

missing bounds check against vma_end

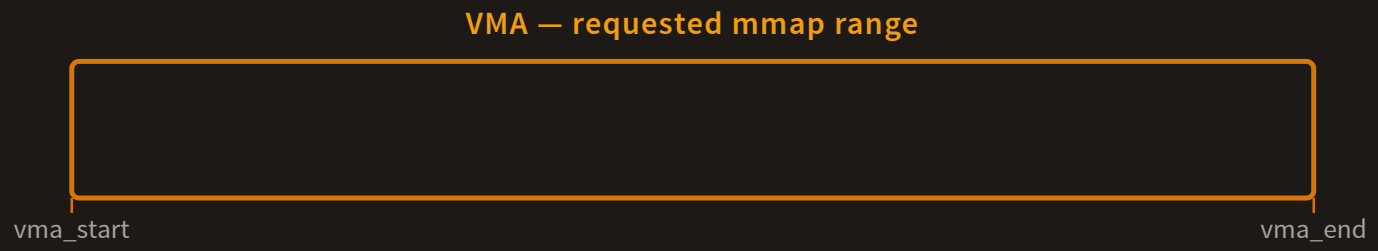


# How To Allocate SGs?



# How To Allocate SGs?

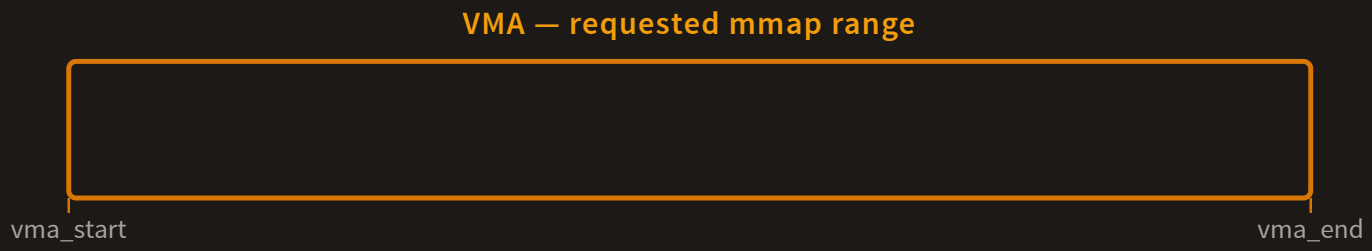
```
static int create_sgt(struct qaic_device *qdev, struct sg_table **sgt_out, u64 size)
{
    struct page **pages;
    int *pages_order;
    int nr_pages = DIV_ROUND_UP(size, PAGE_SIZE);
    int order = MAX_ORDER;









    [...]
}
```



# How To Allocate SGs?

```
static int create_sgt(struct qaic_device *qdev, struct sg_table **sgt_out, u64 size)
{
    struct page **pages;
    int *pages_order;
    int nr_pages = DIV_ROUND_UP(size, PAGE_SIZE);
    int order = MAX_ORDER;

    [...]
}
```

qaic_device: struct for one physical QAIC card

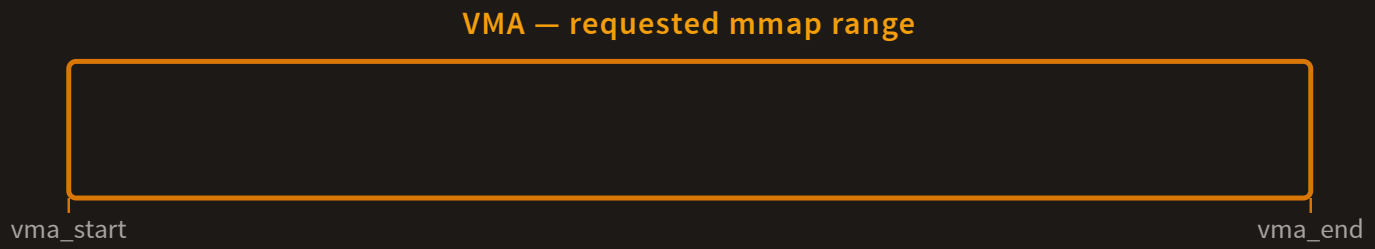


# How To Allocate SGs?

```
static int create_sgt(struct qaic_device *qdev, struct sg_table **sgt_out, u64 size)
{
    struct page **pages;
    int *pages_order;
    int nr_pages = DIV_ROUND_UP(size, PAGE_SIZE);
    int order = MAX_ORDER;

    [...]
}
```

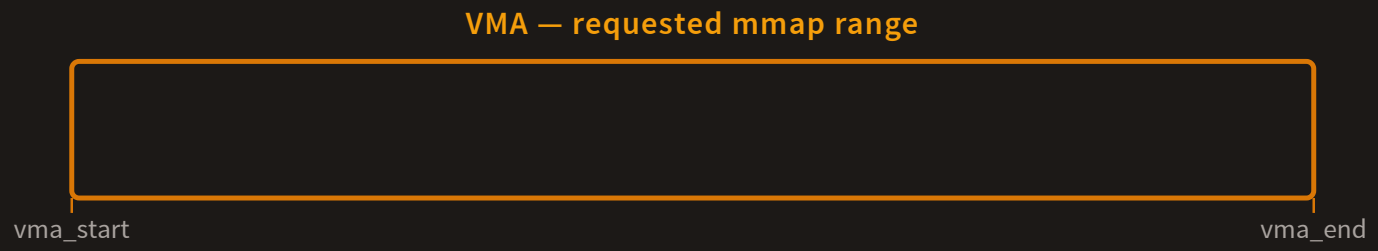
qaic_device: struct for one physical QAIC card

sg_table: the buffer's list of memory chunks



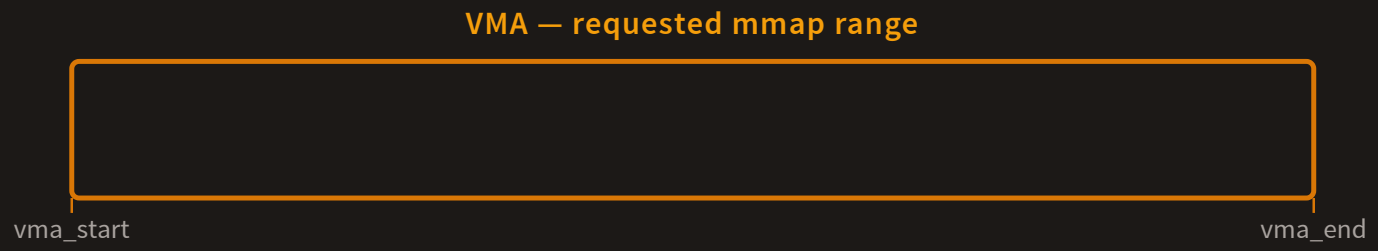
# How To Allocate SGs?

```
static int create_sgt(struct qaic_device *qdev, struct sg_table **sgt_out, u64 size)
{
    struct page **pages;
    int *pages_order;
    int nr_pages = DIV_ROUND_UP(size, PAGE_SIZE);
    int order = MAX_ORDER;

    /* try the biggest page first, fall back smaller on failure -
       every allocation can land at a different order */
    for (i = 0; nr_pages > 0; i++) {
        while (!(pages[i] = alloc_pages(GFP_KERNEL, order)))
            order--;

        pages_order[i] = order;
        nr_pages -= 1 << order;
    }




    [...]
}
```



# How To Allocate SGs?

```
static int create_sgt(struct qaic_device *qdev, struct sg_table **sgt_out, u64 size)
{
    struct page **pages;
    int *pages_order;
    int nr_pages = DIV_ROUND_UP(size, PAGE_SIZE);
    int order = MAX_ORDER;

    /* try the biggest page first, fall back smaller on failure -
       every allocation can land at a different order */
    for (i = 0; nr_pages > 0; i++) {
        while (!(pages[i] = alloc_pages(GFP_KERNEL, order)))
            order--;

        pages_order[i] = order;
        nr_pages -= 1 << order;
    }

    /* one sg entry per allocation, sized by its actual order */
    for (k = 0; k < i; k++, sg = sg_next(sg))
        sg_set_page(sg, pages[k], PAGE_SIZE << pages_order[k], 0);

    [...]
}
```

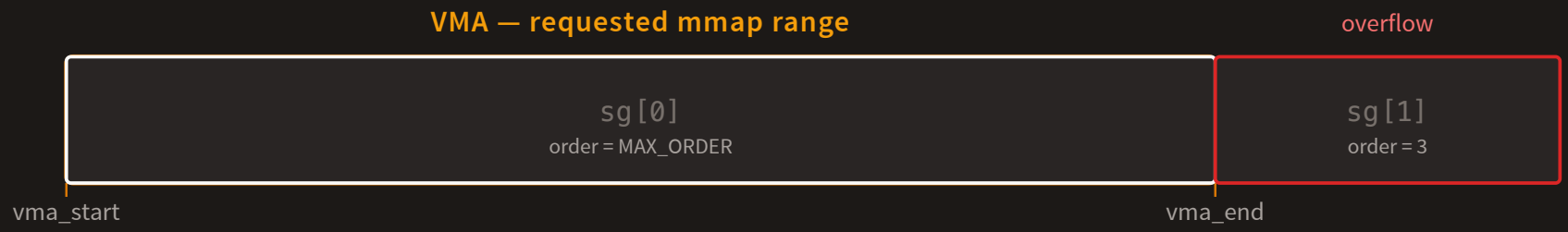


# How To Allocate SGs?

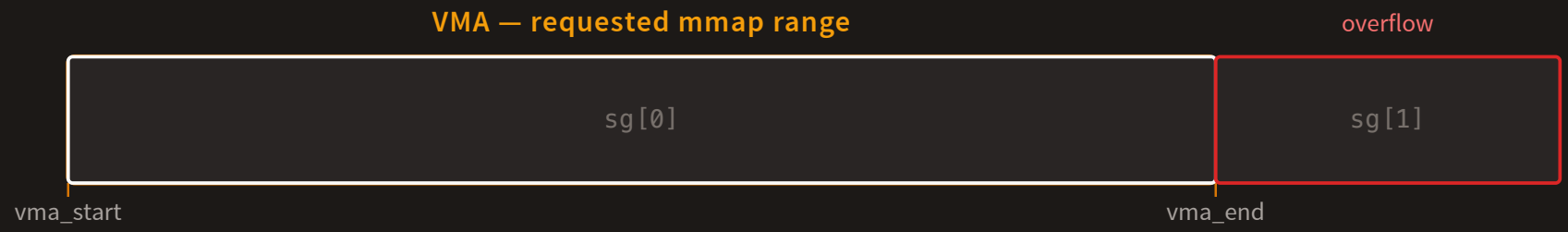
```
static int create_sgt(struct qaic_device *qdev, struct sg_table **sgt_out, u64 size)
{
    struct page **pages;
    int *pages_order;
    int nr_pages = DIV_ROUND_UP(size, PAGE_SIZE);
    int order = MAX_ORDER;

    /* try the biggest page first, fall back smaller on failure -
       every allocation can land at a different order */
    for (i = 0; nr_pages > 0; i++) {
        while (!(pages[i] = alloc_pages(GFP_KERNEL, order)))
            order--;

        pages_order[i] = order;
        nr_pages -= 1 << order;
    }

    /* one sg entry per allocation, sized by its actual order */
    for (k = 0; k < i; k++, sg = sg_next(sg))
        sg_set_page(sg, pages[k], PAGE_SIZE << pages_order[k], 0);

    [...]
}
```

# How To Allocate SGs?

```
static int create_sgt(struct qaic_device *qdev, struct sg_table **sgt_out, u64 size)
{
    struct page **pages;
    int *pages_order;
    int nr_pages = DIV_ROUND_UP(size, PAGE_SIZE);
    int order = MAX_ORDER;

    /* try the biggest page first, fall back smaller on failure -
       every allocation can land at a different order */
    for (i = 0; nr_pages > 0; i++) {
        while (!(pages[i] = alloc_pages(GFP_KERNEL, order)))
            order--;

        pages_order[i] = order;
        nr_pages -= 1 << order;
    }

    /* one sg entry per allocation, sized by its actual order */
    for (k = 0; k < i; k++, sg = sg_next(sg))
        sg_set_page(sg, pages[k], PAGE_SIZE << pages_order[k], 0);

    [...]
}
```

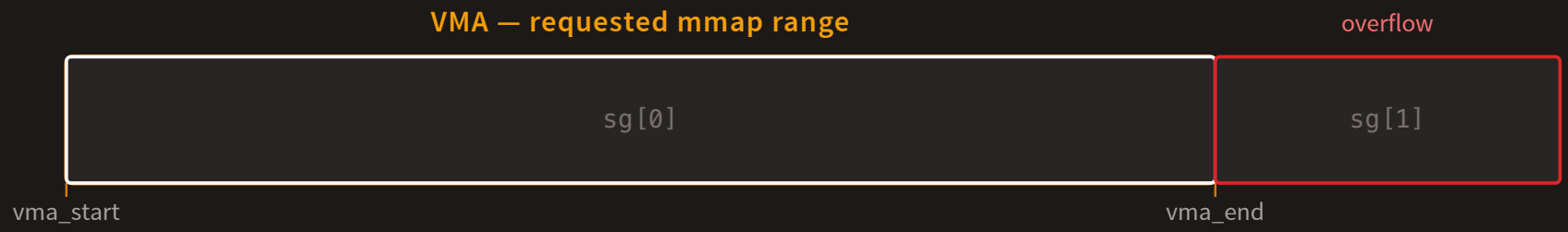


# The Release Part?



# The Release Part?

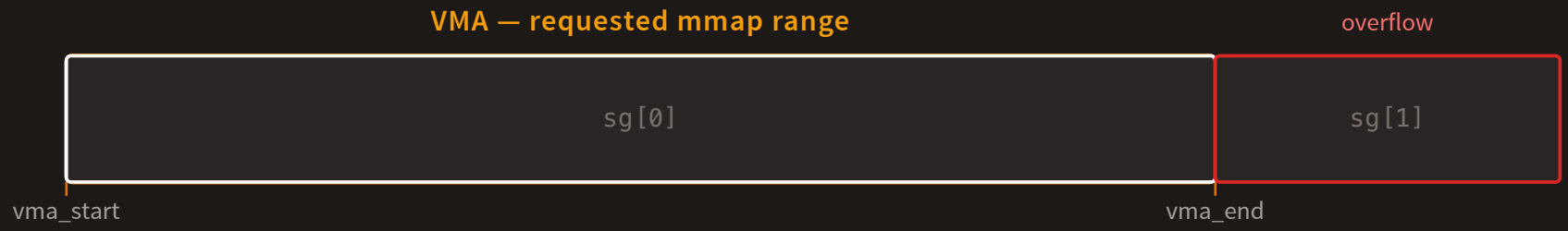
```
/* struct drm_gem_object_funcs - GEM object functions */
struct drm_gem_object_funcs {









};
```



# The Release Part?

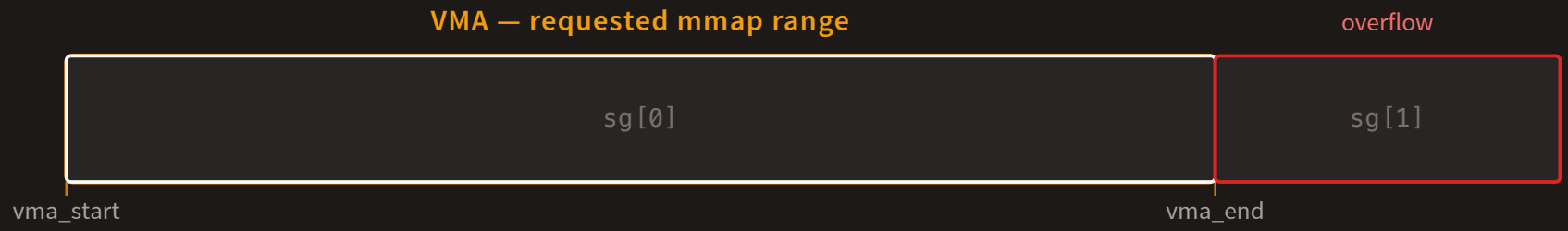
```
/* struct drm_gem_object_funcs - GEM object functions */
struct drm_gem_object_funcs {



};
```

drm_gem_object_funcs: table of GEM lifecycle callbacks



# The Release Part?

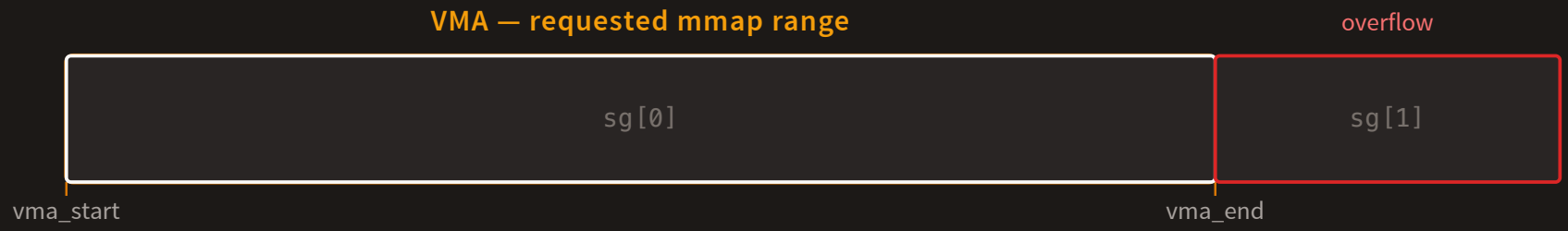
```c
/* struct drm_gem_object_funcs - GEM object functions */
struct drm_gem_object_funcs {
    /* Deconstructor for drm_gem_objects. */
    void (*free)(struct drm_gem_object *obj);

    /* Called upon GEM handle creation. */
    int (*open)(struct drm_gem_object *obj, struct drm_file *file);

    /* Called upon GEM handle release. */
    void (*close)(struct drm_gem_object *obj, struct drm_file *file);

    /* Handle mmap() of the gem object, setup vma accordingly. */
    int (*mmap)(struct drm_gem_object *obj, struct vm_area_struct *vma);

    [...]
};
```



# The Release Part?

```c
/* struct drm_gem_object_funcs - GEM object functions */
struct drm_gem_object_funcs {
    /* Deconstructor for drm_gem_objects. */
    void (*free)(struct drm_gem_object *obj);

    /* Called upon GEM handle creation. */
    int (*open)(struct drm_gem_object *obj, struct drm_file *file);

    /* Called upon GEM handle release. */
    void (*close)(struct drm_gem_object *obj, struct drm_file *file);

    /* Handle mmap() of the gem object, setup vma accordingly. */
    int (*mmap)(struct drm_gem_object *obj, struct vm_area_struct *vma);

    [...]
};
```

```c
/* QAIC's instantiation */
static const struct drm_gem_object_funcs qaic_gem_funcs = {
  .free       = qaic_free_object,
  .print_info = qaic_gem_print_info,
  .mmap       = qaic_gem_object_mmap,
  .vm_ops     = &drm_vm_ops,
};
```



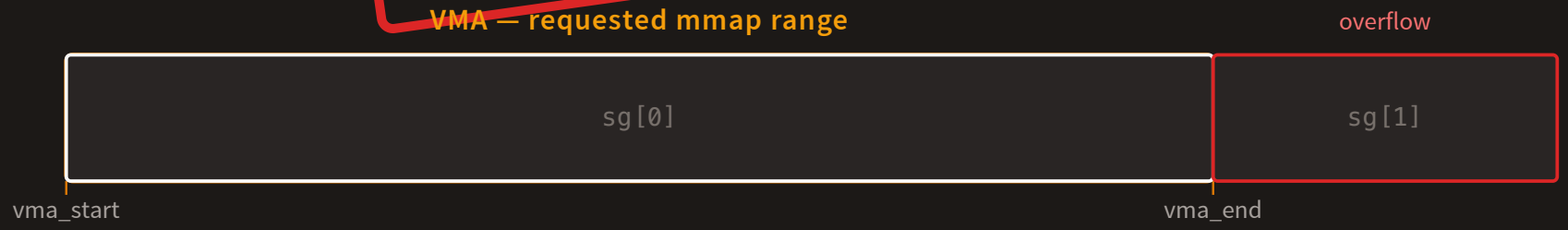
# The Release Part?

```c
/* struct drm_gem_object_funcs - GEM object functions */
struct drm_gem_object_funcs {
    /* Deconstructor for drm_gem_objects. */
    void (*free)(struct drm_gem_object *obj);

    /* Called upon GEM handle creation. */
    int (*open)(struct drm_gem_object *obj, struct drm_file *file);

    /* Called upon GEM handle release. */
    void (*close)(struct drm_gem_object *obj, struct drm_file *file);

    /* Handle mmap() of the gem object, setup vma accordingly. */
    int (*mmap)(struct drm_gem_object *obj, struct vm_area_struct *vma);

    [...]
};
```

```c
/* QAIC's instantiation */
static const struct drm_gem_object_funcs qaic_gem_funcs = {
  .free       = qaic_free_object,
  .print_info = qaic_gem_print_info,
  .mmap       = qaic_gem_object_mmap,
  .vm_ops     = &drm_vm_ops,
};
```

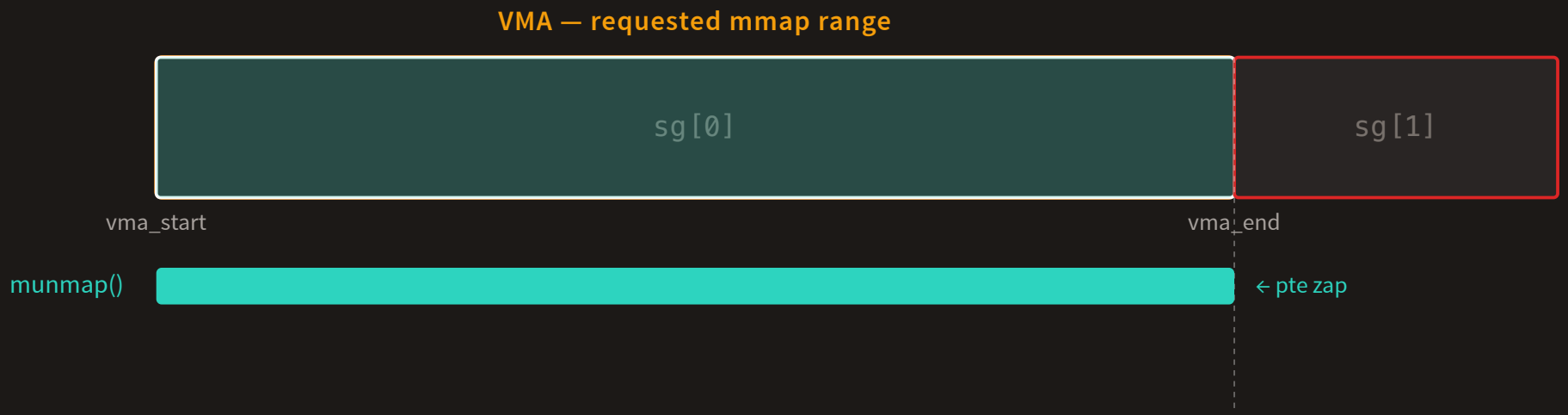
**no dedicated release**

**→ only Linux's generic one**

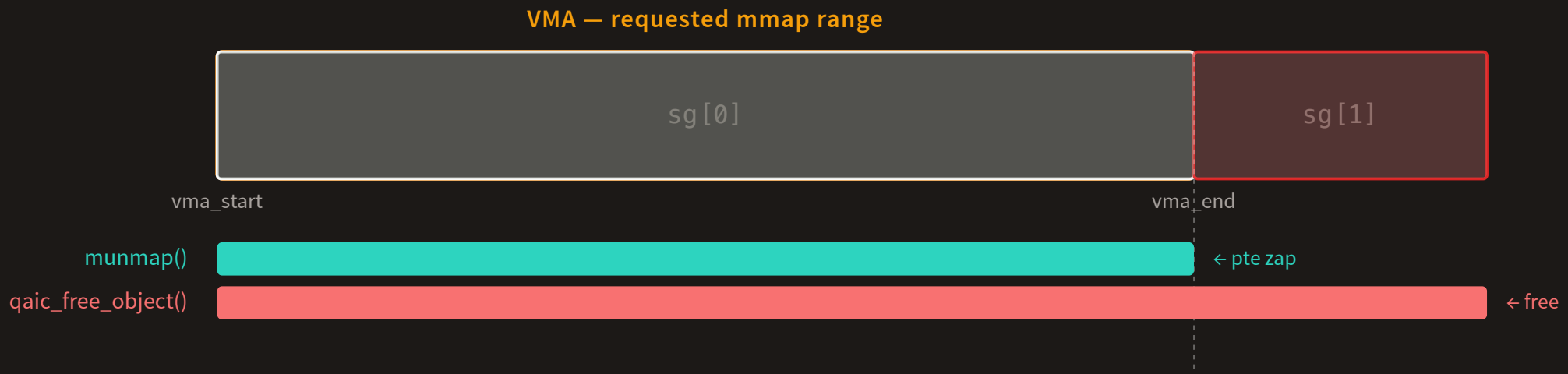


# The Generic Release

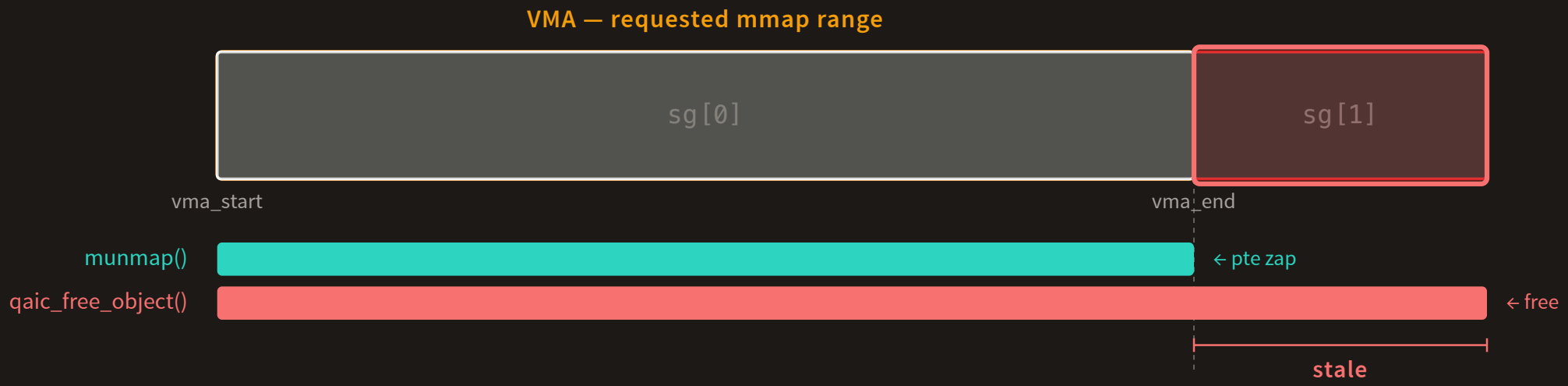
# The Generic Release



# The Generic Release



# The Generic Release



# Exploitation

# Three Phases to Root

# Three Phases to Root

# Three Phases to Root

**Capability:** stale mapping of freed physical memory

# Three Phases to Root

**Capability:** stale mapping of freed physical memory

- Reused for any operation

# Three Phases to Root

**Capability:** stale mapping of freed physical memory

- Reused for any operation
- Arbitrary read and write access

# Three Phases to Root

**Capability:** stale mapping of freed physical memory

- Reused for any operation
- Arbitrary read and write access

**01**

**Reclamation**

`reclaim the freed page`

# Three Phases to Root

**Capability:** stale mapping of freed physical memory

- Reused for any operation
- Arbitrary read and write access

**01**

**Reclamation**

`reclaim the freed page`

**02**

**Primitive**

`pipe_buffer → arbitrary R/W`

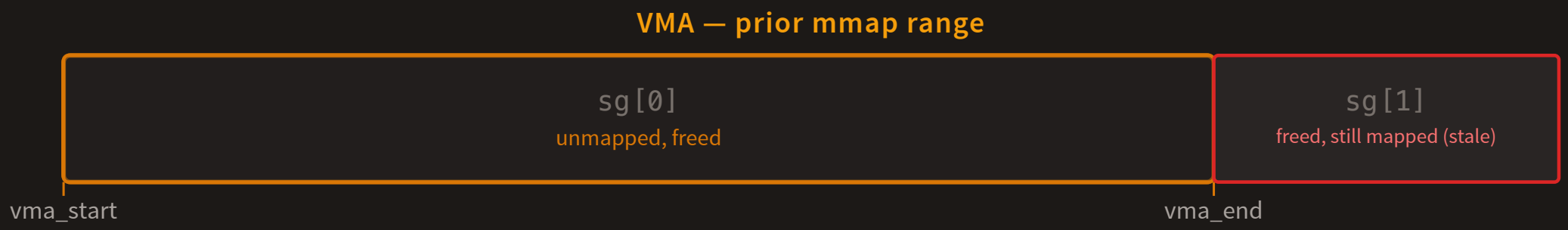
# Three Phases to Root

**Capability:** stale mapping of freed physical memory

- Reused for any operation
- Arbitrary read and write access

| 01 | 02 | 03 |
|---|---|---|
| **Reclamation** | **Primitive** | **Privesc** |
| `reclaim the freed page` | `pipe_buffer → arbitrary R/W` | `arbitrary R/W → root` |

# The Stale Page Mapping



# The Stale Page Mapping

stale

freed

# The Stale Page Mapping

**Goal:** reclaim the order-3 page

stale

freed

# The Stale Page Mapping

**Goal:** reclaim the order-3 page

**Question:** as what?

# The Stale Page Mapping

**Goal:** reclaim the order-3 page

**Question:** as what?

- page table

stale

freed

# The Stale Page Mapping

**Goal:** reclaim the order-3 page

**Question:** as what?

- page table
- `pipe_buffer` slab page

# The Stale Page Mapping

**Goal:** reclaim the order-3 page

**Question:** as what?

- page table
- `pipe_buffer` slab page
- ...

stale

freed

# The Stale Page Mapping

**Goal:** reclaim the order-3 page

**Question:** as what?

- page table
- `pipe_buffer` slab page
- ...

**Constraint:** kernel exploitation must be near 100% reliable to be weaponizable

# The Stale Page Mapping

**Goal:** reclaim the order-3 page

**Question:** as what?

- page table
- `pipe_buffer` slab page
- ...

**Constraint:** kernel exploitation must be near 100% reliable to be weaponizable

- `sk_buff->frag` data page

# Reclaiming with `sk_buff` Fragments

stale

freed

# Reclaiming with `sk_buff` Fragments

```
struct sk_buff {
    ...
    unsigned int truesize;
    ...
    struct skb_shared_info *end;
};
```

stale

freed

# Reclaiming with `sk_buff` Fragments

```
struct sk_buff {
    ...
    unsigned int truesize;
    ...
    struct skb_shared_info *end;
};
```

sk_buff: kernel's descriptor for one network packet

stale

freed

# Reclaiming with `sk_buff` Fragments

```
struct sk_buff {
    ...
    unsigned int truesize;
    ...
    struct skb_shared_info *end;
};
```

```
struct skb_shared_info {
    ...
    unsigned char nr_frags;
    skb_frag_t frags[MAX_SKB_FRAGS];
};
```

# Reclaiming with `sk_buff` Fragments

```
struct sk_buff {
    ...
    unsigned int truesize;
    ...
    struct skb_shared_info *end;
};
```

```
struct skb_shared_info {
    ...
    unsigned char nr_frags;
    skb_frag_t frags[MAX_SKB_FRAGS];
};
```

frags: array of pages backing this fragment

stale

freed

# Reclaiming with `sk_buff` Fragments

```
struct sk_buff {
    ...
    unsigned int truesize;
    ...
    struct skb_shared_info *end;
};
```

```
struct skb_shared_info {
    ...
    unsigned char nr_frags;
    skb_frag_t frags[MAX_SKB_FRAGS];
};
```

```
struct sk_buff *alloc_skb_with_frags(..., int order, ...)
{




}
```

**stale**

freed

# Reclaiming with `sk_buff` Fragments

```
struct sk_buff {
    ...
    unsigned int truesize;
    ...
    struct skb_shared_info *end;
};
```

```
struct skb_shared_info {
    ...
    unsigned char nr_frags;
    skb_frag_t frags[MAX_SKB_FRAGS];
};
```

```
struct sk_buff *alloc_skb_with_frags(..., int order, ...)
{
                                        alloc_skb_with_frags: allocates frag pages
}
```

**stale**

freed

# Reclaiming with `sk_buff` Fragments

```
struct sk_buff {
    ...
    unsigned int truesize;
    ...
    struct skb_shared_info *end;
};
```

```
struct skb_shared_info {
    ...
    unsigned char nr_frags;
    skb_frag_t frags[MAX_SKB_FRAGS];
};
```

```
struct sk_buff *alloc_skb_with_frags(..., int order, ...)
{
    [...]

    while (data_len) {
        while (order && PAGE_ALIGN(data_len) < (PAGE_SIZE << order))
            order--;                    // shrink to fit what's left

        page = alloc_pages(gfp_mask, order); // frag page allocation

        [...]
    }
}
```

**stale**

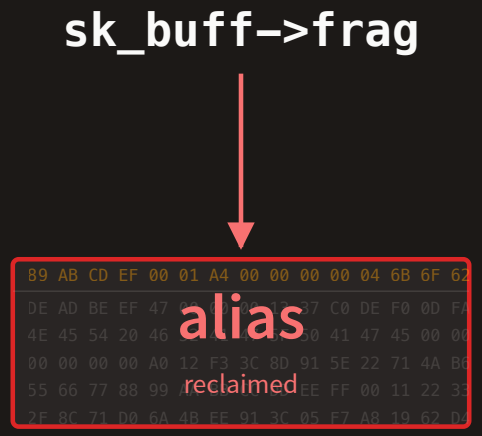
freed

# Reclaiming with `sk_buff` Fragments

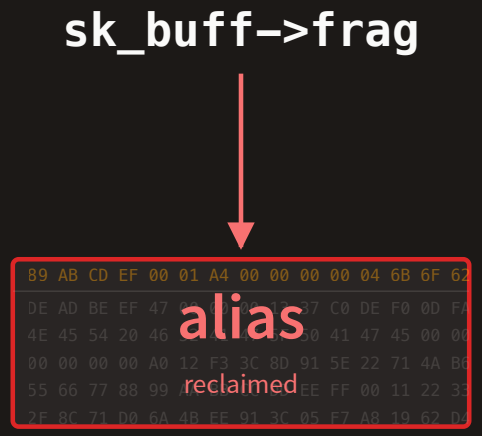
```
struct sk_buff {
    ...
    unsigned int truesize;
    ...
    struct skb_shared_info *end;
};
```

```
struct skb_shared_info {
    ...
    unsigned char nr_frags;
    skb_frag_t frags[MAX_SKB_FRAGS];
};
```

```
struct sk_buff *alloc_skb_with_frags(..., int order, ...)
{
    [...]

    while (data_len) {
        while (order && PAGE_ALIGN(data_len) < (PAGE_SIZE << order))
            order--;                    // shrink to fit what's left

        page = alloc_pages(gfp_mask, order); // frag page allocation

        [...]
    }
}
```

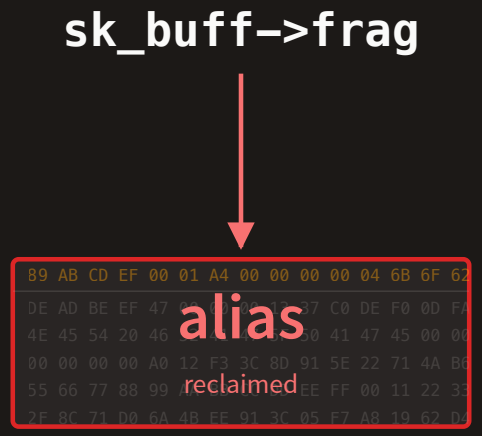


# Confirming the Reclaim



# Confirming the Reclaim

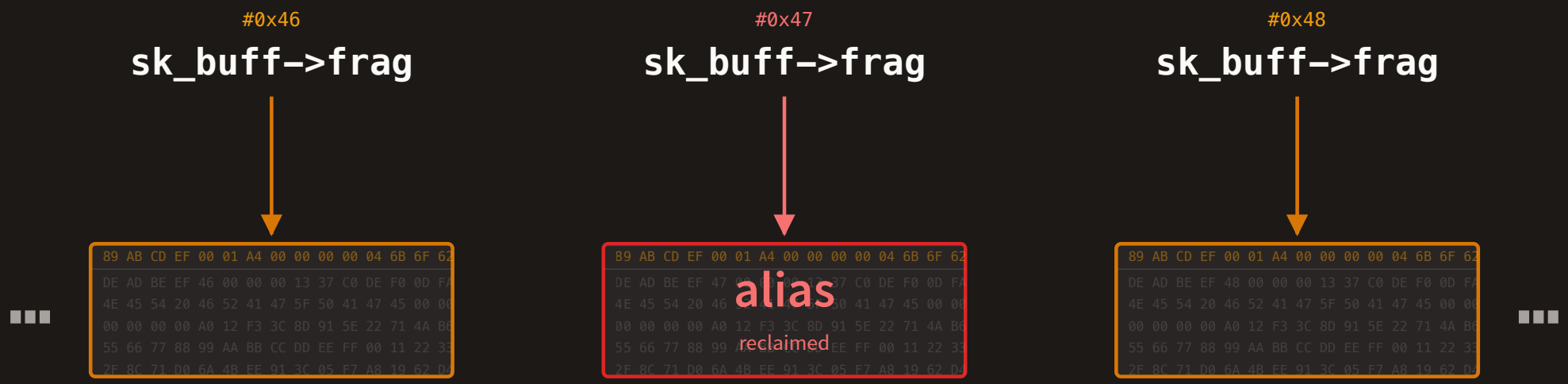
**Spray:** `sk_buff->frag` data pages



# Confirming the Reclaim

**Spray:** `sk_buff->frag` data pages

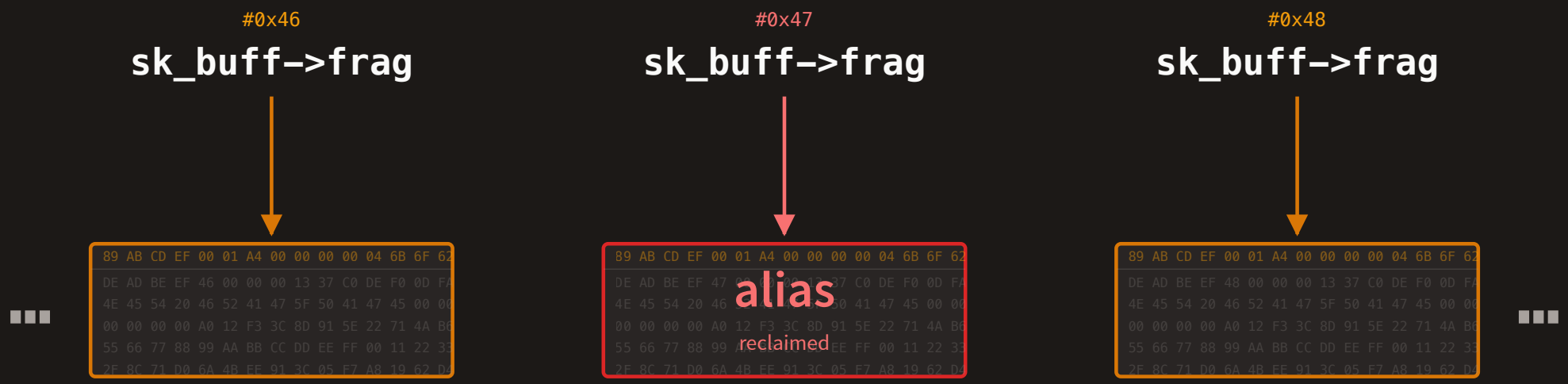
with unique cookie `DE AD BE EF XX`



# Confirming the Reclaim

**Spray:** `sk_buff->frag` data pages

- with unique cookie `DE AD BE EF XX`
- where `XX` is the `sk_buff` identifier

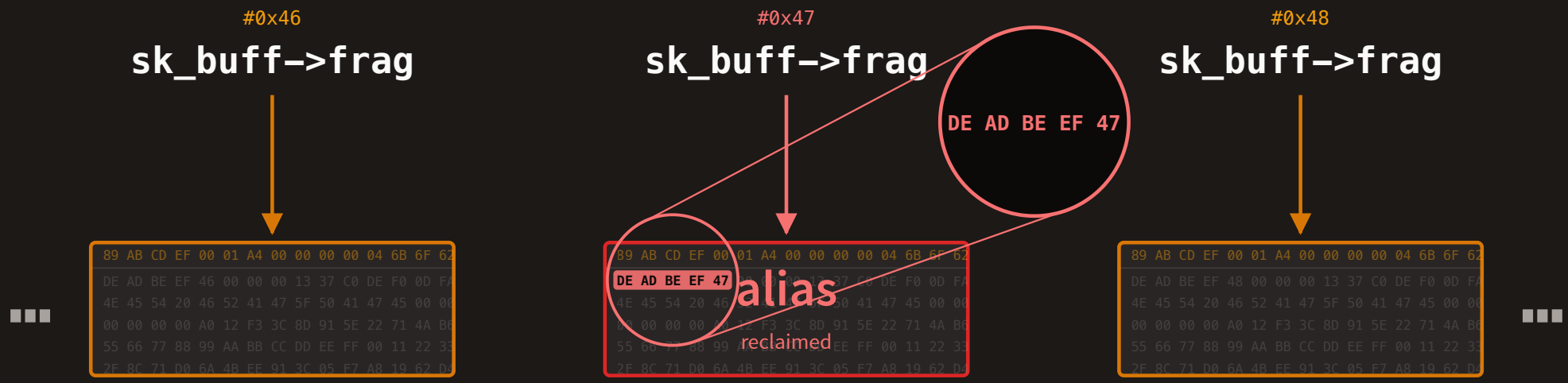


# Confirming the Reclaim

**Spray:** `sk_buff->frag` data pages

- with unique cookie `DE AD BE EF XX`
- where `XX` is the `sk_buff` identifier

**Confirm:** read cookie via alias



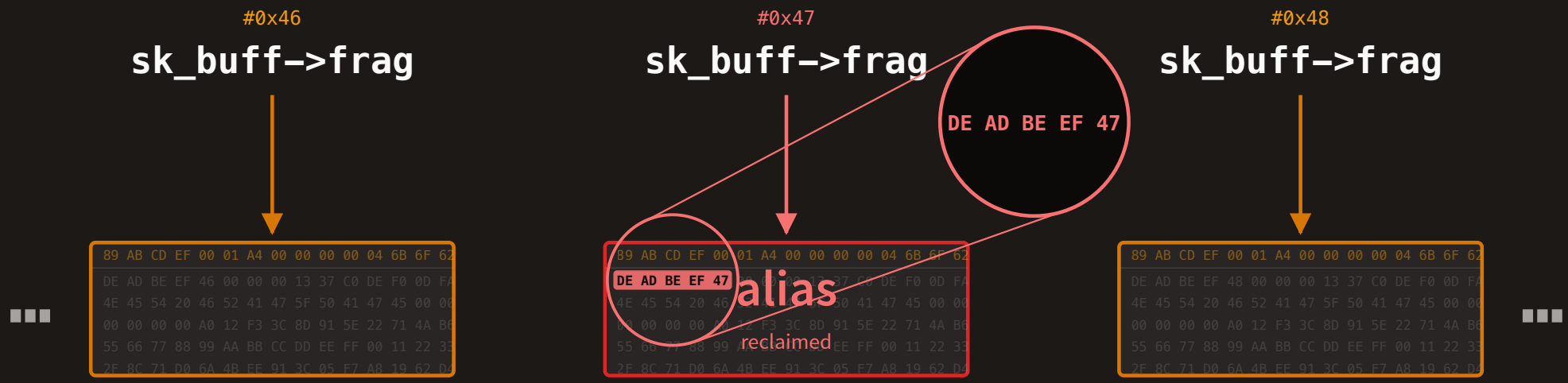
# Confirming the Reclaim

**Spray:** `sk_buff->frag` data pages

- with unique cookie `DE AD BE EF XX`
- where `XX` is the `sk_buff` identifier

**Confirm:** read cookie via alias

- `DE AD BE EF 47` means `sk_buff` number `0x47`



# Confirming the Reclaim

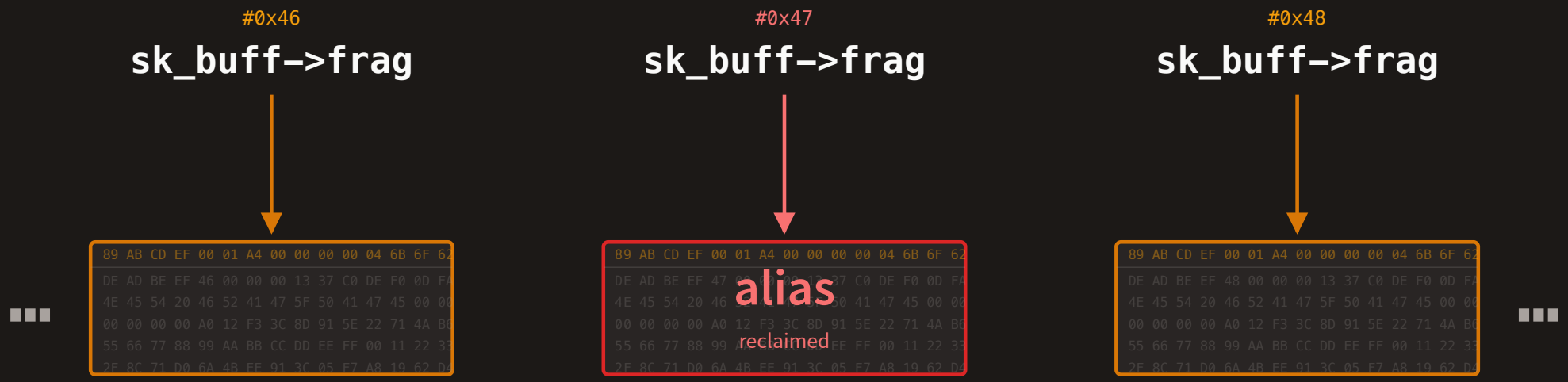
**Spray:** `sk_buff->frag` data pages

- with unique cookie `DE AD BE EF XX`
- where `XX` is the `sk_buff` identifier

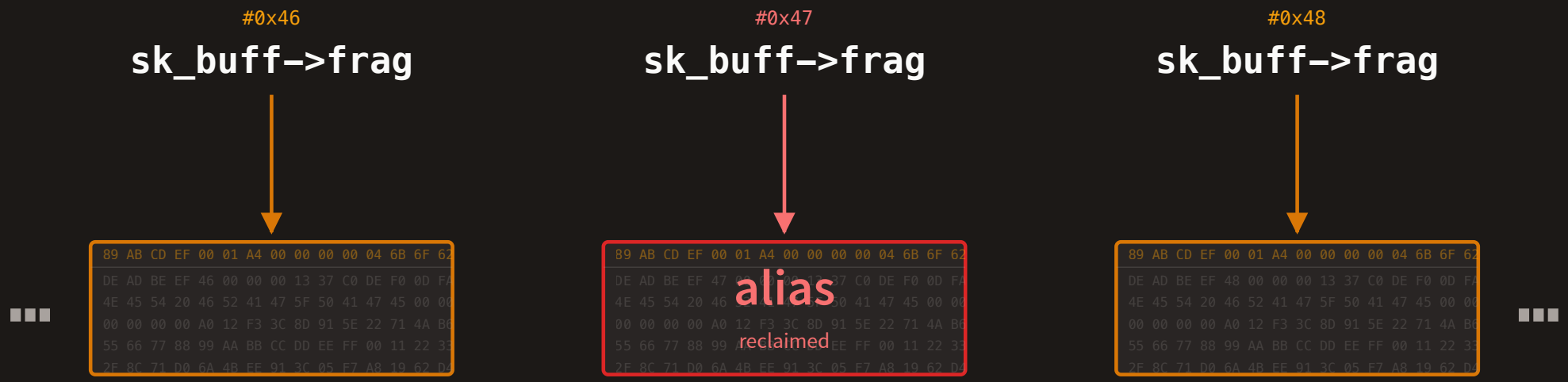
**Confirm:** read cookie via alias

- `DE AD BE EF 47` means `sk_buff` number `0x47`

**Result:** reliable reclamation

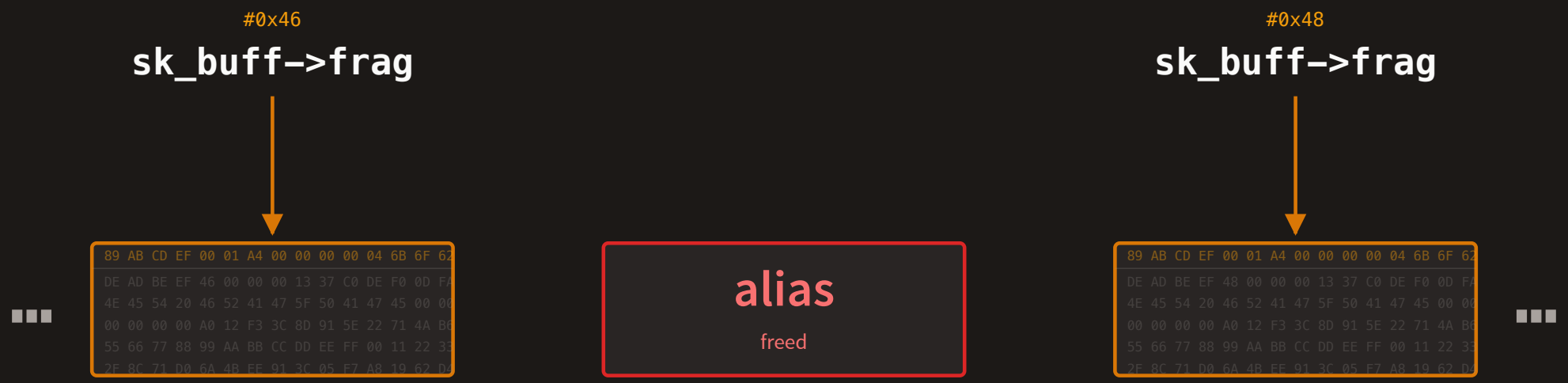


# Free and Reclaim The Second



# Free and Reclaim The Second

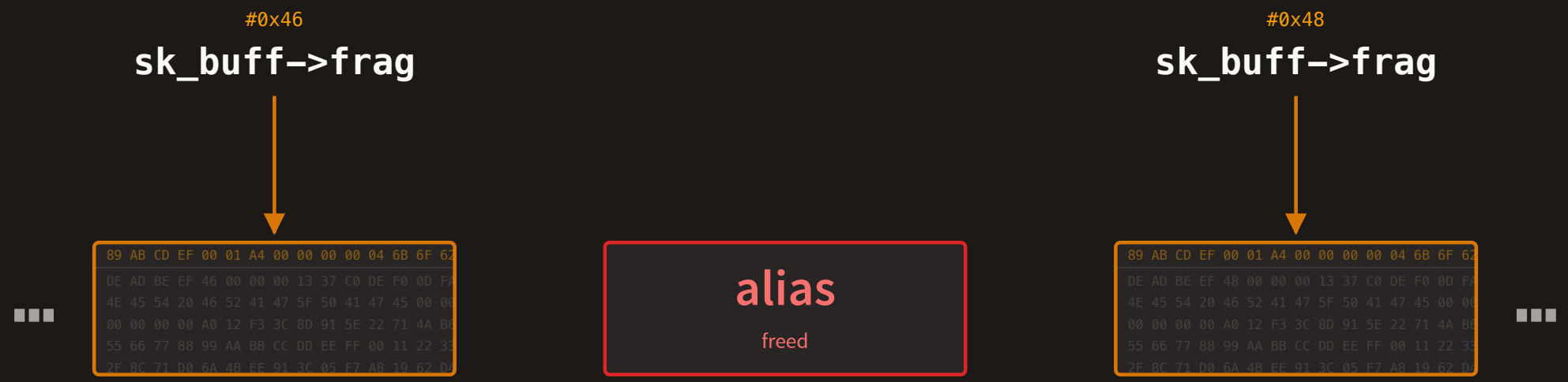
Free: `__free_pages(sk_buff->frag)` — `#0x47`



# Free and Reclaim The Second

**Free:** `__free_pages(sk_buff->frag)` — `#0x47`

releases the order-3 page again



# Free and Reclaim The Second

**Free:** `__free_pages(sk_buff->frag)` — `#0x47`

releases the order-3 page again

**Reclaim:** as `pipe_buffer` slab page

# Free and Reclaim The Second

**Free:** `__free_pages(sk_buff->frag)` — `#0x47`

- releases the order-3 page again

**Reclaim:** as `pipe_buffer` slab page

- each pipe holds a ring of `pipe_buffer` structs, one per queued page

# Free and Reclaim The Second

**Free:** `__free_pages(sk_buff->frag)` — `#0x47`

- releases the order-3 page again

**Reclaim:** as `pipe_buffer` slab page

- each pipe holds a ring of `pipe_buffer` structs, one per queued page
- ring size is tunable from userspace via `fcntl(F_SETPIPE_SZ)`

# Free and Reclaim The Second

**Free:** `__free_pages(sk_buff->frag)` — `#0x47`

- releases the order-3 page again

**Reclaim:** as `pipe_buffer` slab page

- each pipe holds a ring of `pipe_buffer` structs, one per queued page
- ring size is tunable from userspace via `fcntl(F_SETPIPE_SZ)`
- elastic object: `pipe_buffer[16]` rounds up into `kmalloc-1k`, backed by order-3 slabs

# Free and Reclaim The Second

**Free:** `__free_pages(sk_buff->frag)` — `#0x47`

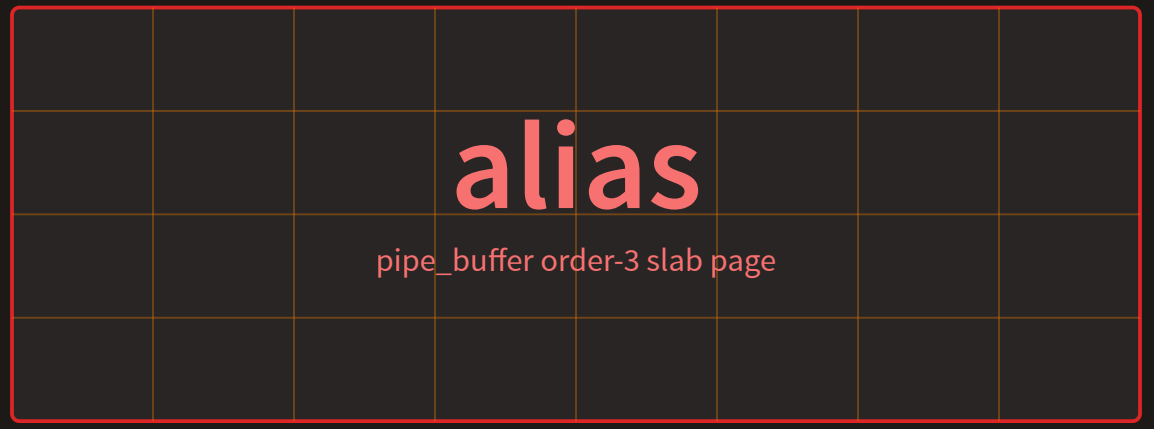
- releases the order-3 page again

**Reclaim:** as `pipe_buffer` slab page

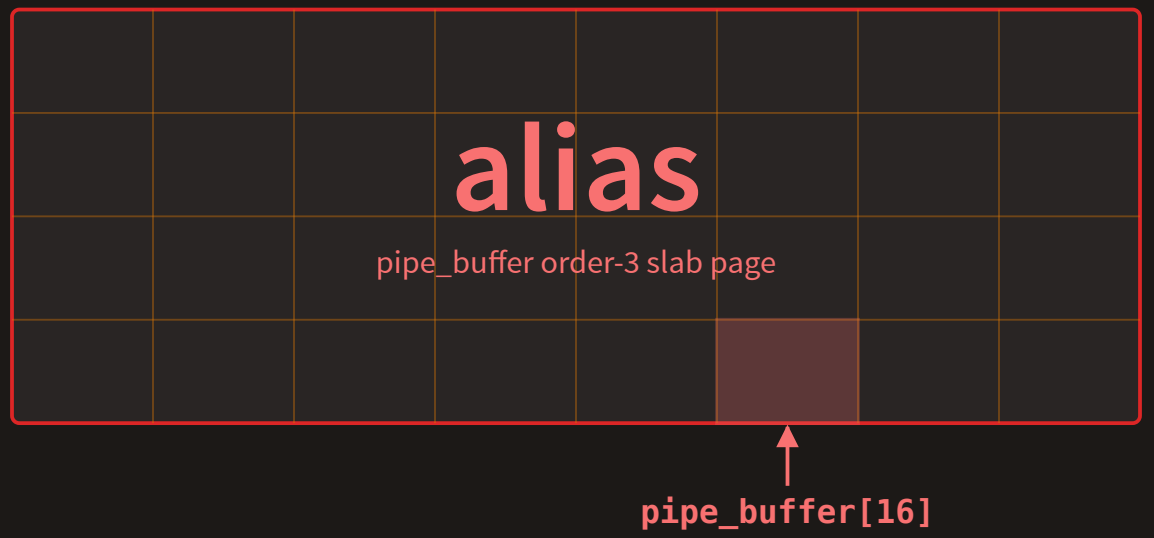
- each pipe holds a ring of `pipe_buffer` structs, one per queued page
- ring size is tunable from userspace via `fcntl(F_SETPIPE_SZ)`
- elastic object: `pipe_buffer[16]` rounds up into `kmalloc-1k`, backed by order-3 slabs

**Result:** reliable reclamation

# Why `pipe_buffer`?



# Why `pipe_buffer` ?



# Why `pipe_buffer`?

```
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {




};
```

# Why `pipe_buffer`?

```
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;




};
```

# Why `pipe_buffer`?

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;



};
```

# Why `pipe_buffer`?

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;


};
```

# Why `pipe_buffer`?

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

# Why `pipe_buffer` ?

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

Strong leak primitive:

# Why `pipe_buffer` ?

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

**Strong leak primitive:**

- `page` — leaks the **vmemmap** base
  vmemmap_base = 1GB-aligned `pipe_buffer->page`

# Why `pipe_buffer` ?

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

**Strong leak primitive:**

- `page` — leaks the **vmemmap** base
  vmemmap_base = 1GB-aligned `pipe_buffer->page`
- `ops` — leaks a **kernel code** base
  kernel_base = `pipe_buffer->ops` − anon_pipe_buf_ops_offset

# Why `pipe_buffer` ?

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

**Strong leak primitive:**

- `page` — leaks the **vmemmap** base
  vmemmap_base = 1GB-aligned `pipe_buffer->page`
- `ops` — leaks a **kernel code** base
  kernel_base = `pipe_buffer->ops` − anon_pipe_buf_ops_offset

**Strong write primitive:**

# Why `pipe_buffer` ?

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

## Strong leak primitive:

- `page` — leaks the **vmemmap** base
  vmemmap_base = 1GB-aligned `pipe_buffer->page`
- `ops` — leaks a **kernel code** base
  kernel_base = `pipe_buffer->ops` − anon_pipe_buf_ops_offset

## Strong write primitive:

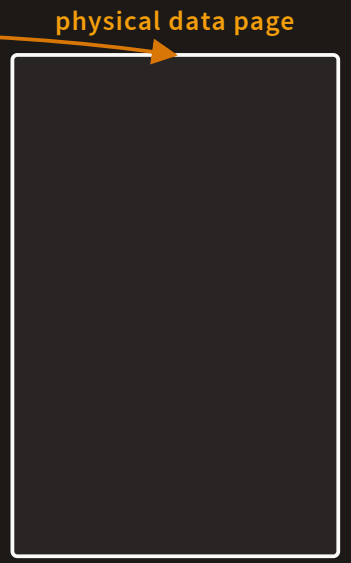
- `page` can be redirected to **arbitrary** pages
  turns read()/write() into an arbitrary-page R/W primitive

# Forging `pipe_buffer` → Arbitrary Phys R/W

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

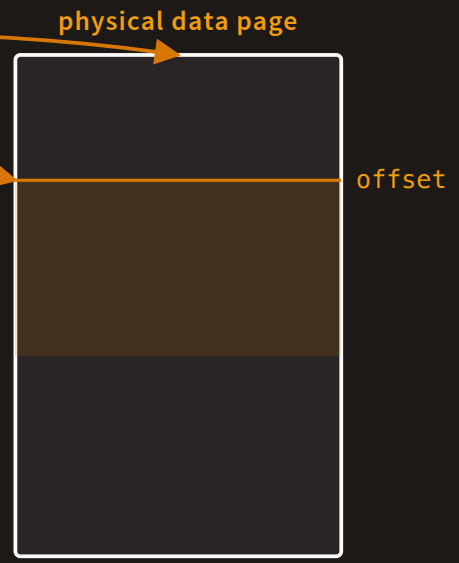
# Forging `pipe_buffer` → Arbitrary Phys R/W

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



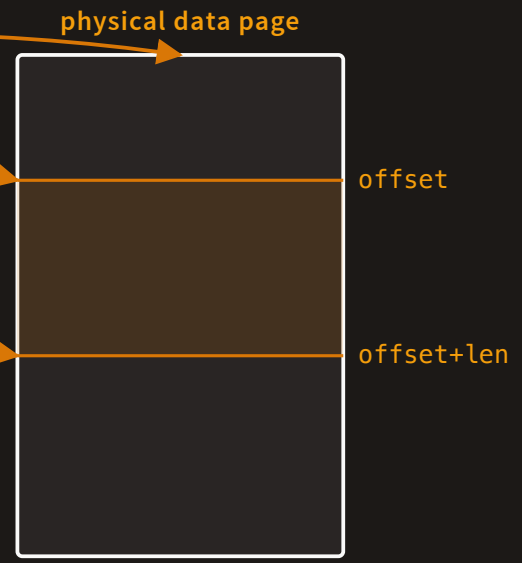
# Forging `pipe_buffer` → Arbitrary Phys R/W

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



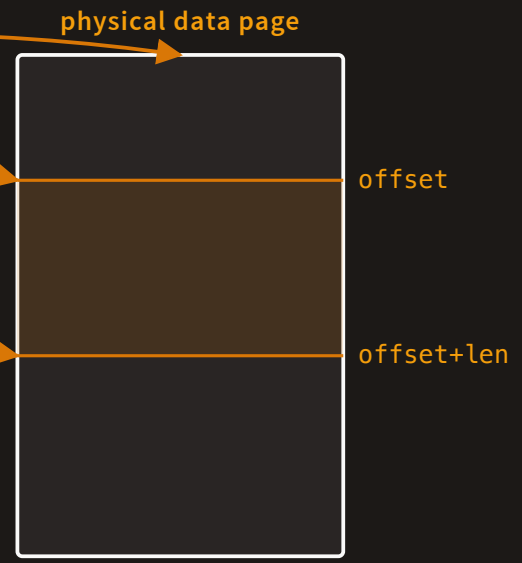
# Forging `pipe_buffer` → Arbitrary Phys R/W

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



# Forging `pipe_buffer` → Arbitrary Phys R/W

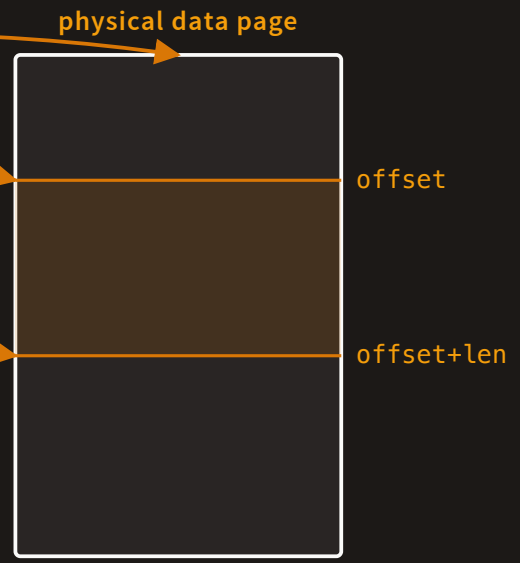
```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```




Normal behaviour:

# Forging `pipe_buffer` → Arbitrary Phys R/W

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

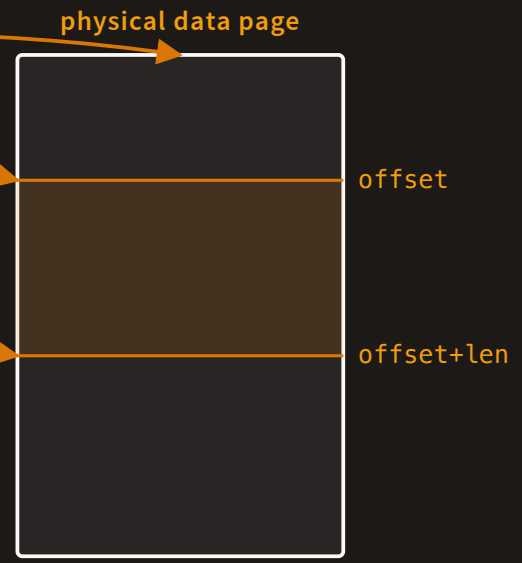



**Normal behaviour:**

- `read()` reads bytes from the physical data page at `offset`

# Forging `pipe_buffer` → Arbitrary Phys R/W

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```




**Normal behaviour:**

- `read()` reads bytes from the physical data page at `offset`
- `write()` writes bytes to the physical data page at `offset+len`

# Forging `pipe_buffer` → Arbitrary Phys R/W

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```


**Normal behaviour:**

- `read()` reads bytes from the physical data page at `offset`
- `write()` writes bytes to the physical data page at `offset+len`

**Attacker scenario:**

# Forging `pipe_buffer` → Arbitrary Phys R/W

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```


**Normal behaviour:**

- `read()` reads bytes from the physical data page at `offset`
- `write()` writes bytes to the physical data page at `offset+len`

**Attacker scenario:**

- use the alias to control `page`, `offset`, `len`
- referring to an arbitrary attacker-selected physical page

# Forging `pipe_buffer` → Arbitrary Phys R/W

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```


**Normal behaviour:**

- `read()` reads bytes from the physical data page at `offset`
- `write()` writes bytes to the physical data page at `offset+len`

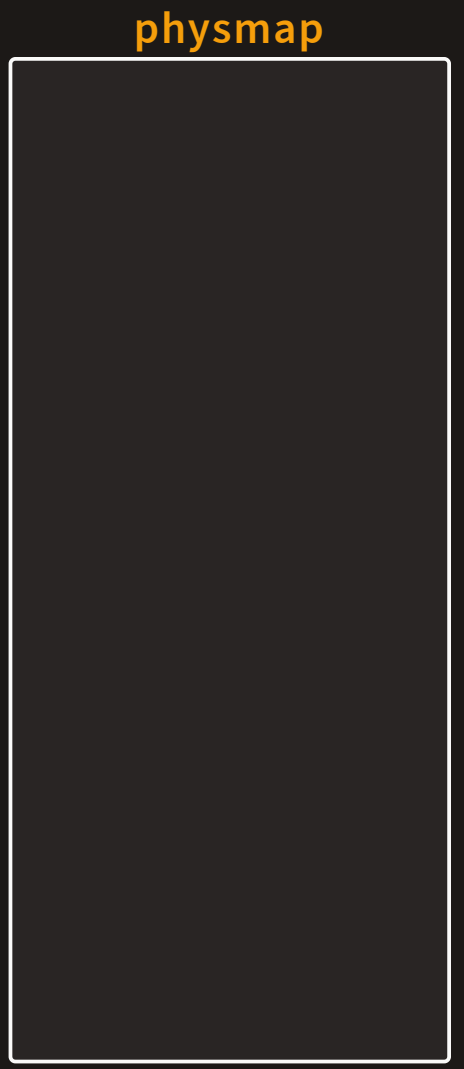
**Attacker scenario:**

- use the alias to control `page`, `offset`, `len`
- referring to an arbitrary attacker-selected physical page

**Result:** arb phys r/w

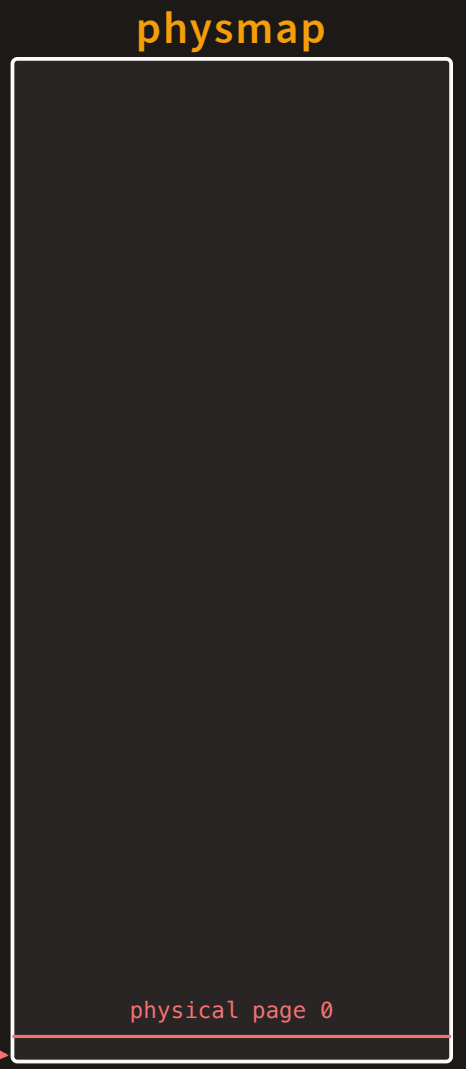
# Sanity Check: Physical Page 0

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



# Sanity Check: Physical Page 0

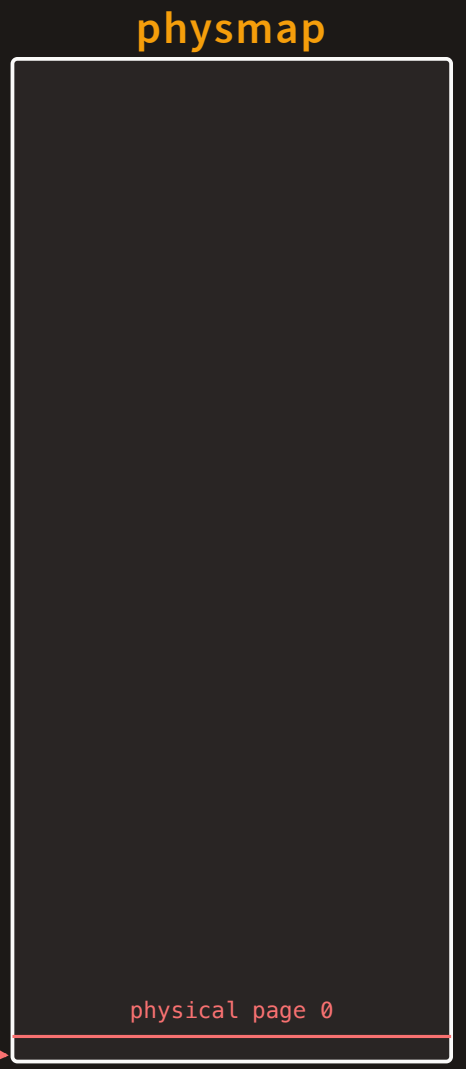
```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



# Sanity Check: Physical Page 0

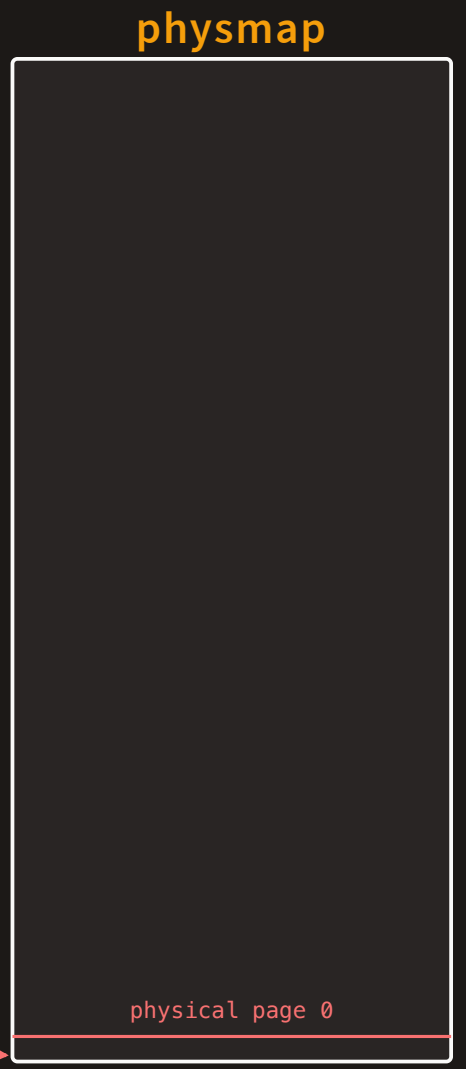
```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

Sanity check:



# Sanity Check: Physical Page 0

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```




**Sanity check:**

- check page 0 for an ELF header

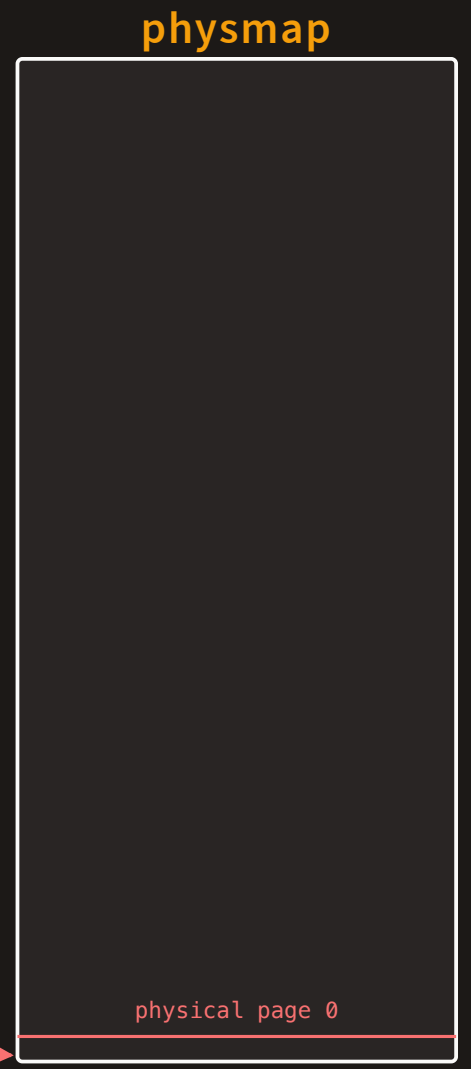
# Sanity Check: Physical Page 0

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

**Sanity check:**

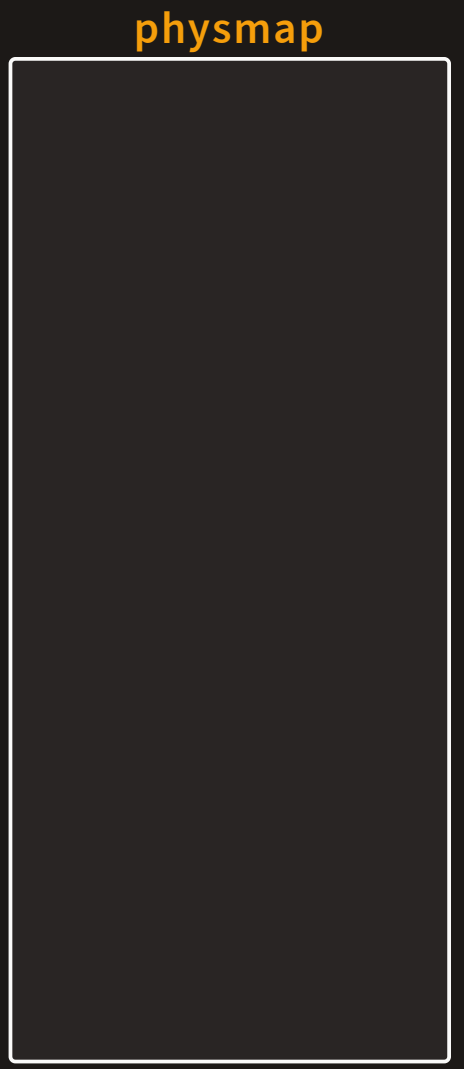
- check page 0 for an ELF header

**Result:** confirmed and working arb phys r/w



# Finding the Kernel

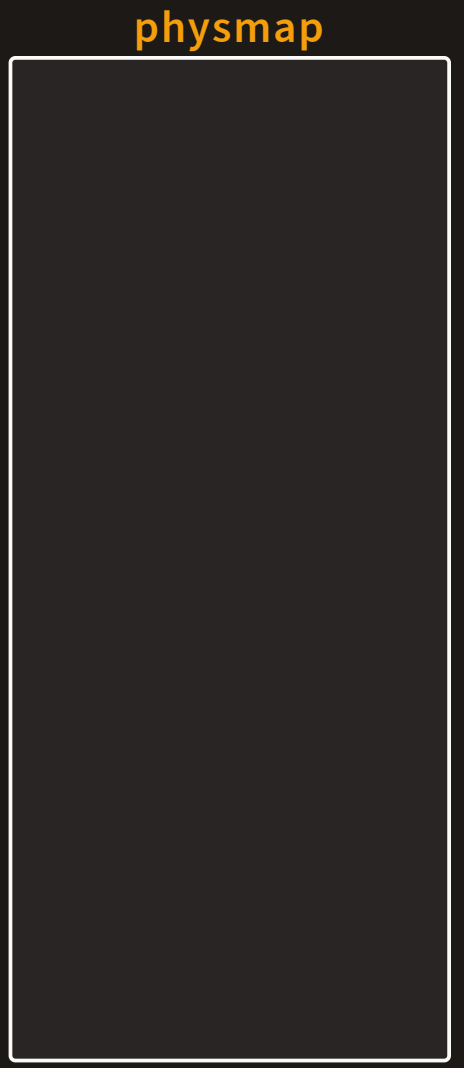
```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



# Finding the Kernel

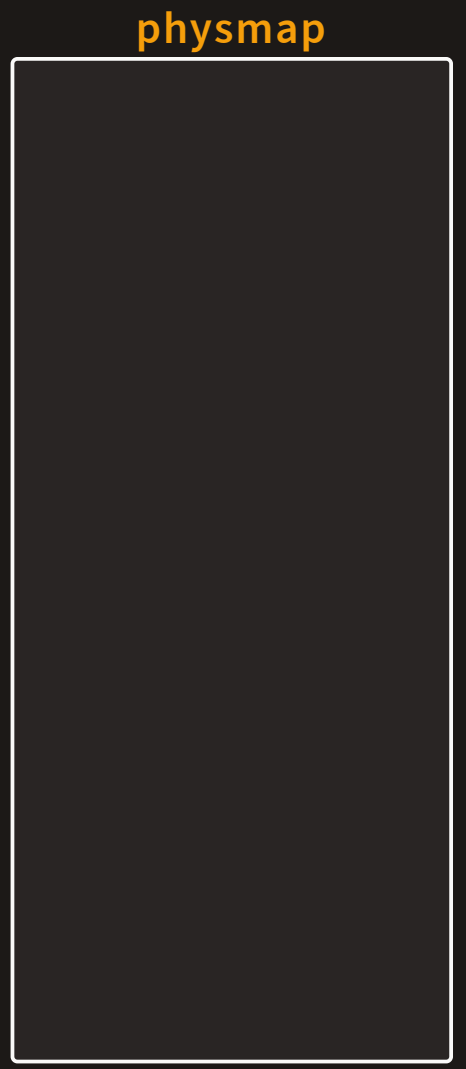
```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

Calculating phys kernel base:



# Finding the Kernel

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

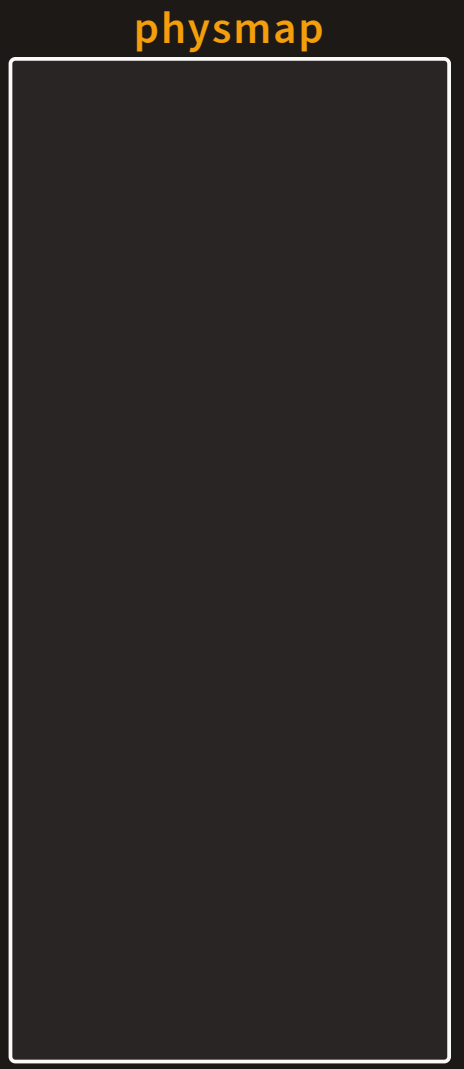



## Calculating phys kernel base:

- on most Android devices the kernel's physical load address ( `LOAD_PA` ) is fixed by the bootloader/device config
  when known, this skips the scan entirely and gives the kernel base directly

# Finding the Kernel

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



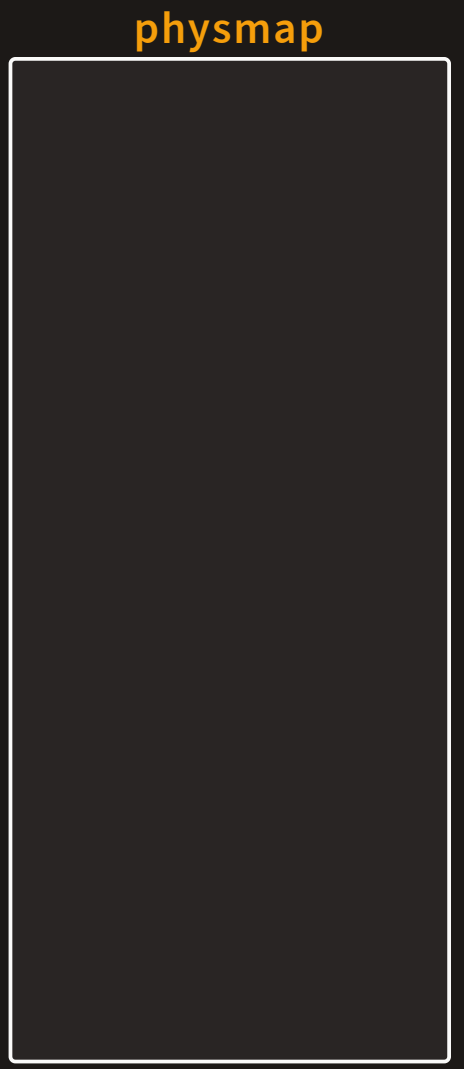

## Calculating phys kernel base:

- on most Android devices the kernel's physical load address ( `LOAD_PA` ) is fixed by the bootloader/device config
  when known, this skips the scan entirely and gives the kernel base directly

## Finding phys kernel base:

# Finding the Kernel

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```




## Calculating phys kernel base:

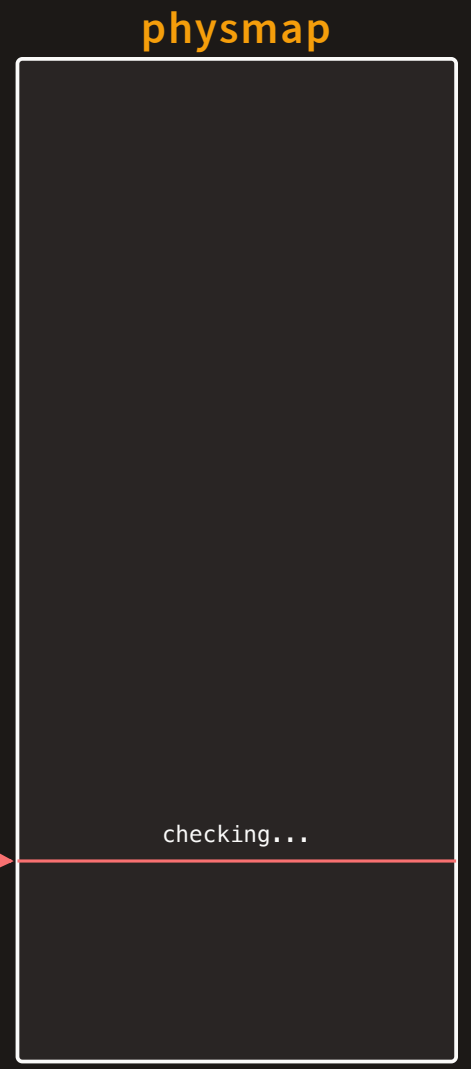
- on most Android devices the kernel's physical load address ( `LOAD_PA` ) is fixed by the bootloader/device config
  when known, this skips the scan entirely and gives the kernel base directly

## Finding phys kernel base:

- without `LOAD_PA` , I scan the physmap upward
  read each candidate back and check for a recognizable kernel structure

# Finding the Kernel

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```




## Calculating phys kernel base:

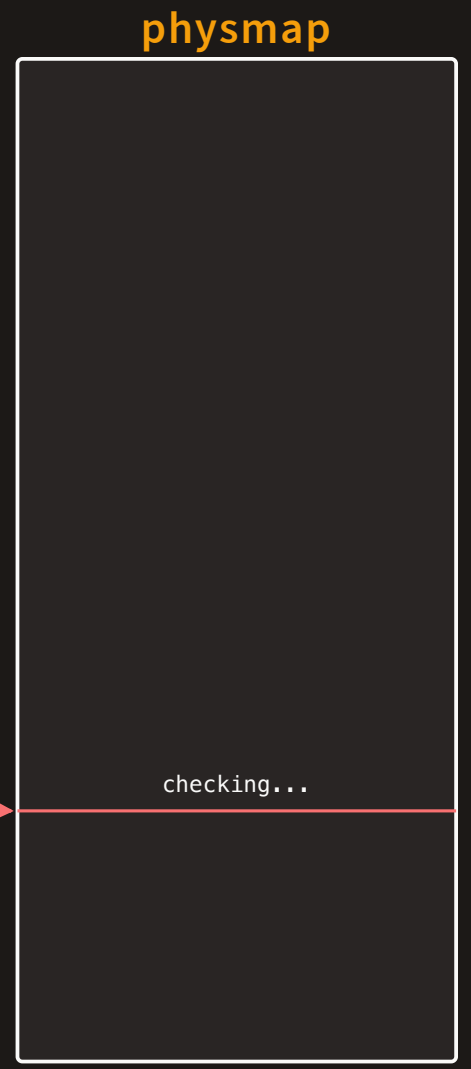
- on most Android devices the kernel's physical load address ( `LOAD_PA` ) is fixed by the bootloader/device config
  when known, this skips the scan entirely and gives the kernel base directly

## Finding phys kernel base:

- without `LOAD_PA` , I scan the physmap upward
  read each candidate back and check for a recognizable kernel structure

# Finding the Kernel

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```




## Calculating phys kernel base:

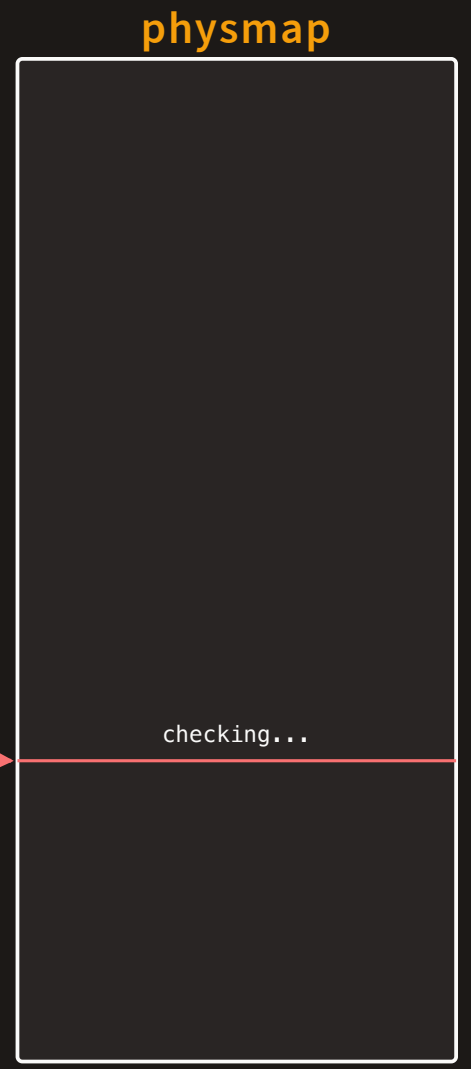
- on most Android devices the kernel's physical load address ( `LOAD_PA` ) is fixed by the bootloader/device config
  when known, this skips the scan entirely and gives the kernel base directly

## Finding phys kernel base:

- without `LOAD_PA` , I scan the physmap upward
  read each candidate back and check for a recognizable kernel structure

# Finding the Kernel

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```




## Calculating phys kernel base:

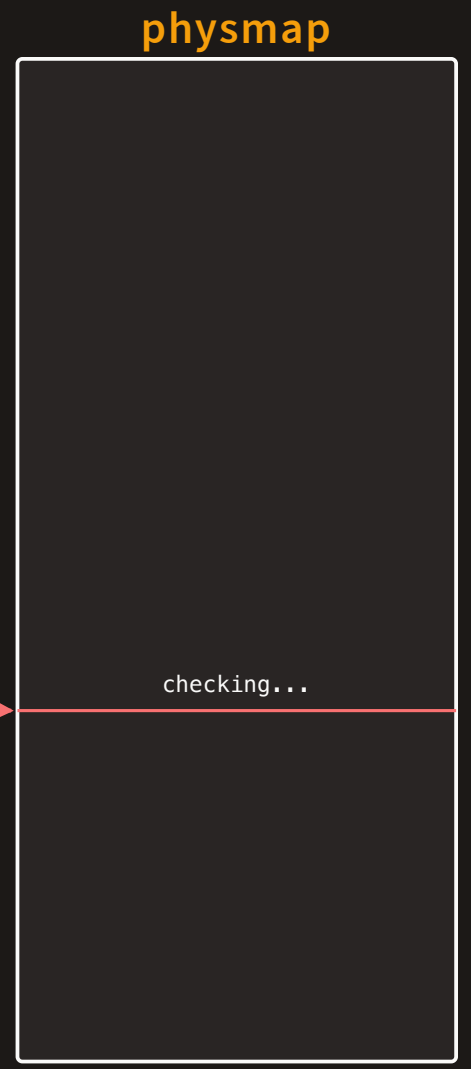
- on most Android devices the kernel's physical load address ( `LOAD_PA` ) is fixed by the bootloader/device config
  - when known, this skips the scan entirely and gives the kernel base directly

## Finding phys kernel base:

- without `LOAD_PA` , I scan the physmap upward
  - read each candidate back and check for a recognizable kernel structure

# Finding the Kernel

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```




## Calculating phys kernel base:

- on most Android devices the kernel's physical load address ( `LOAD_PA` ) is fixed by the bootloader/device config
  - when known, this skips the scan entirely and gives the kernel base directly

## Finding phys kernel base:

- without `LOAD_PA` , I scan the physmap upward
  - read each candidate back and check for a recognizable kernel structure

# Finding the Kernel

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```


## Calculating phys kernel base:

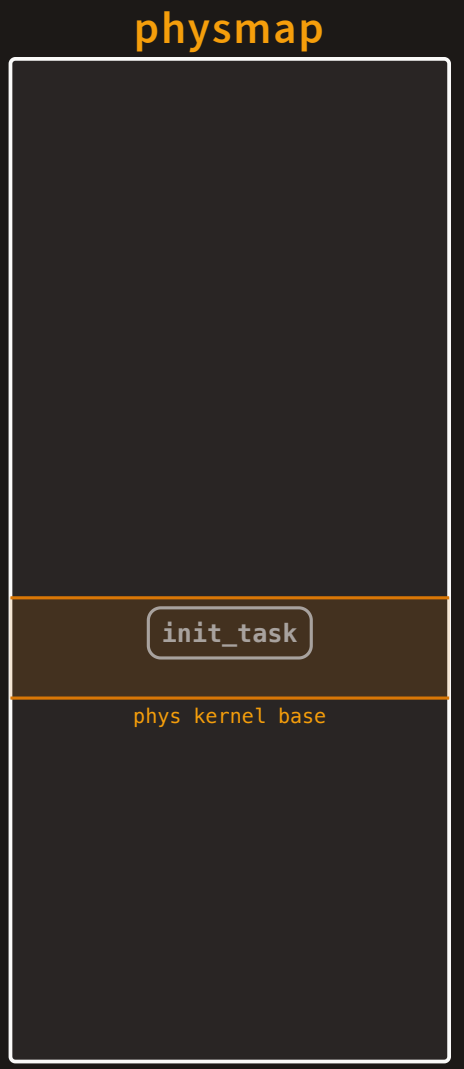
- on most Android devices the kernel's physical load address ( `LOAD_PA` ) is fixed by the bootloader/device config
  - when known, this skips the scan entirely and gives the kernel base directly

## Finding phys kernel base:

- without `LOAD_PA` , I scan the physmap upward
  - read each candidate back and check for a recognizable kernel structure

# Locating `current` Task

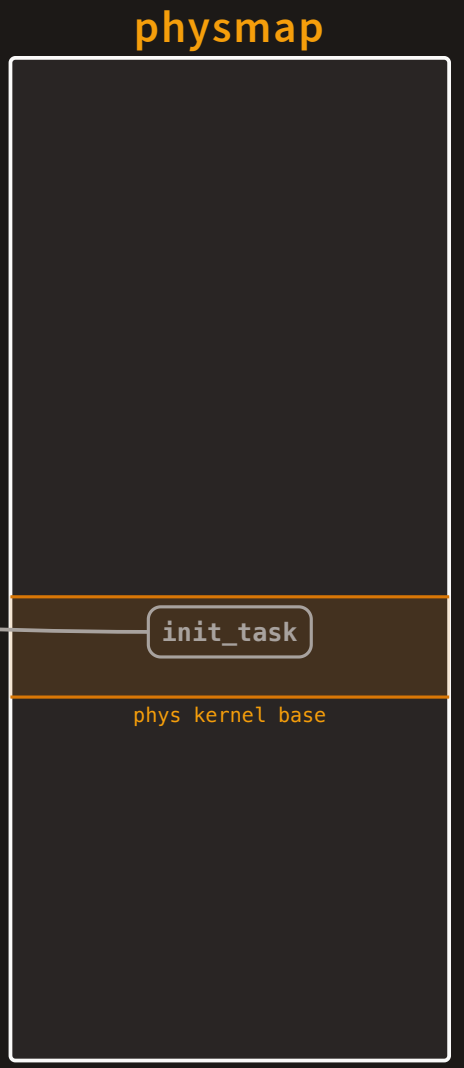
```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

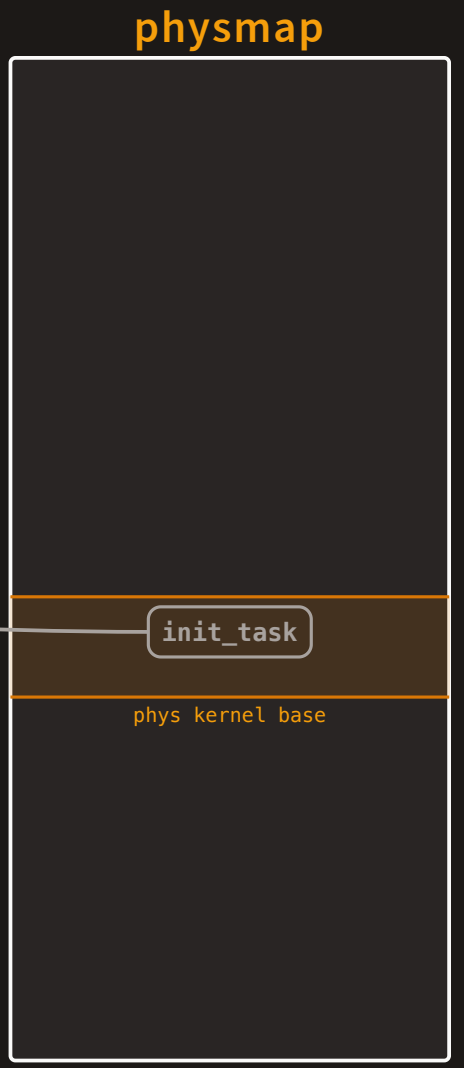
```c
struct task_struct {



};
```



# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

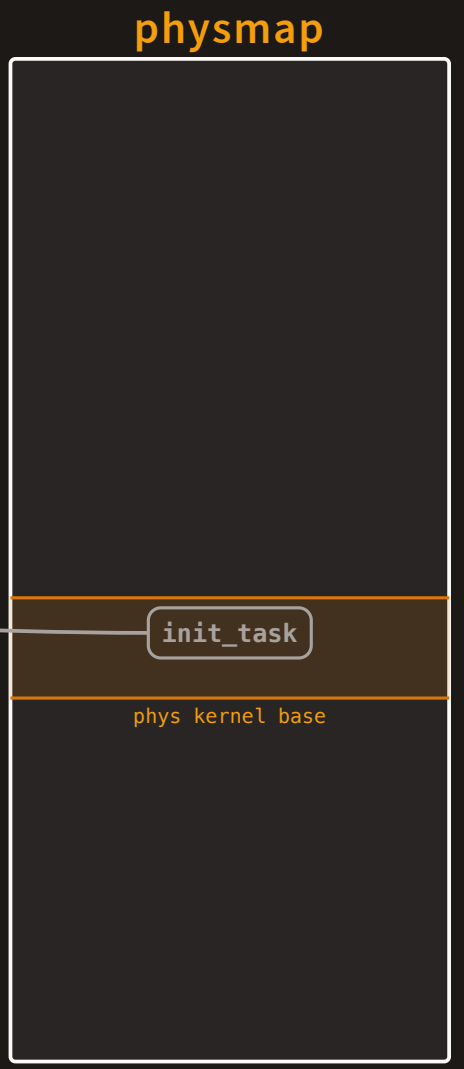
```c
struct task_struct {
    [...]
    // executable name, e.g. "swapper/0"
    char comm[TASK_COMM_LEN];



};
```



# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

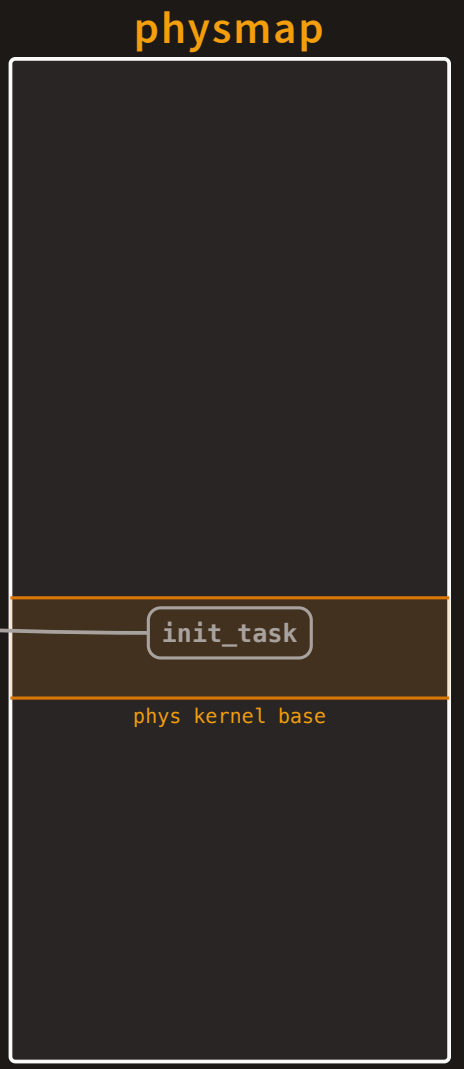
```c
struct task_struct {
    [...]
    // executable name, e.g. "swapper/0"
    char comm[TASK_COMM_LEN];
    [...]
    // linked into the global process list
    struct list_head tasks;



};
```



# Locating `current` Task

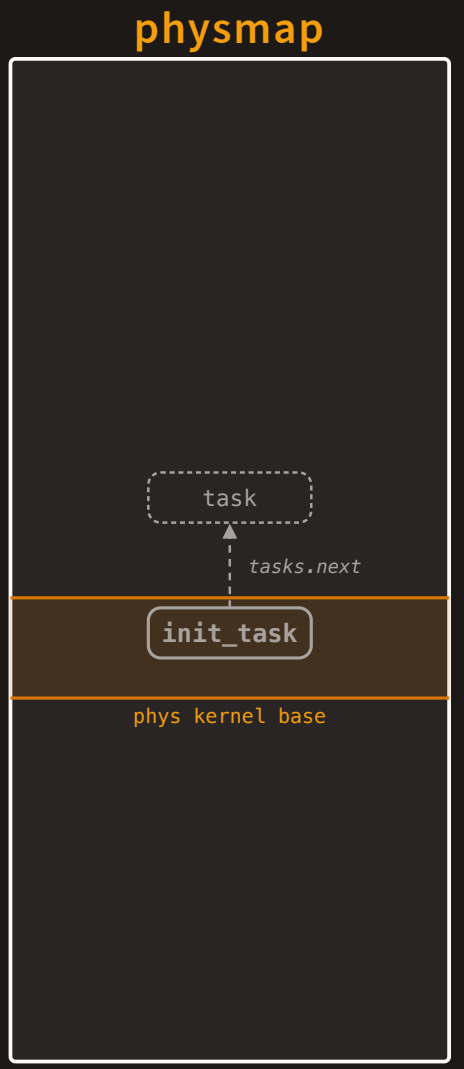
```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

```c
struct task_struct {
    [...]
    // executable name, e.g. "swapper/0"
    char comm[TASK_COMM_LEN];
    [...]
    // linked into the global process list
    struct list_head tasks;
    [...]
    // effective (overridable) subjective task credentials
    const struct cred __rcu *cred;
    [...]
};
```



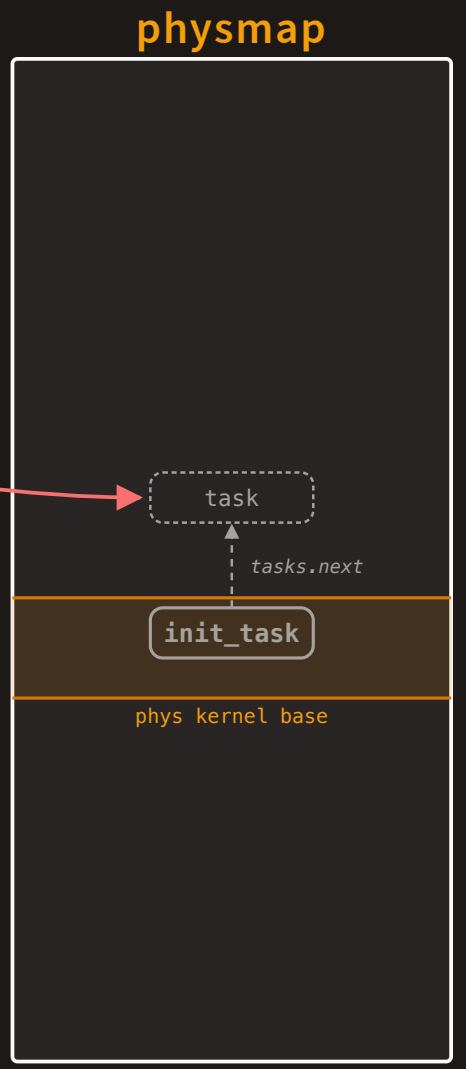
# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



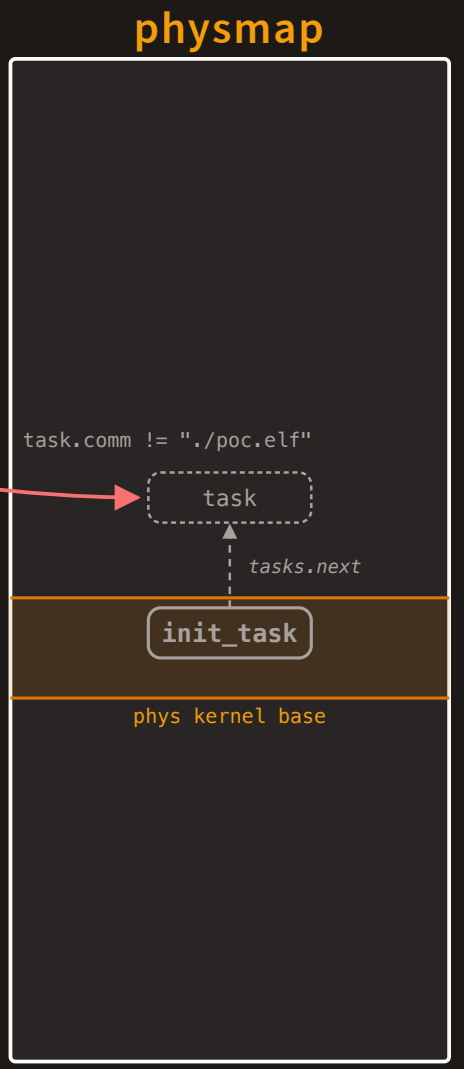
# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



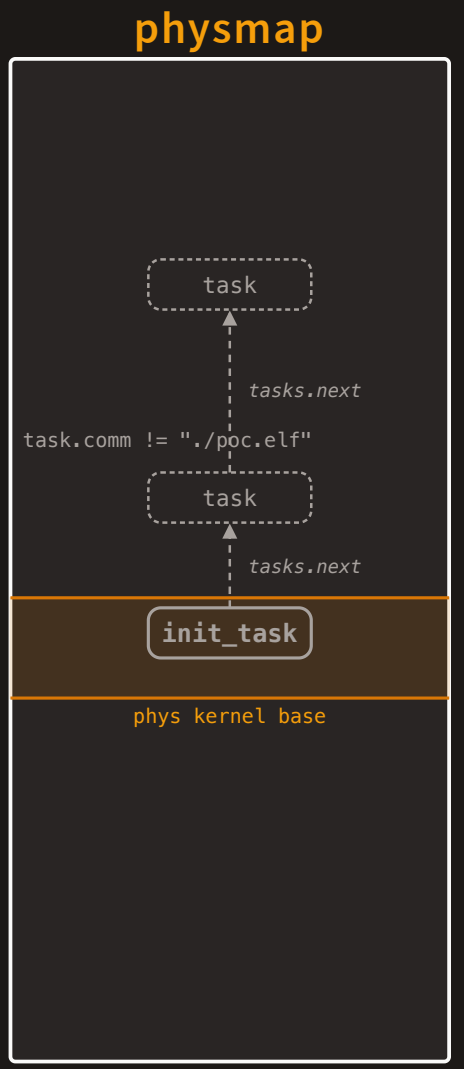
# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



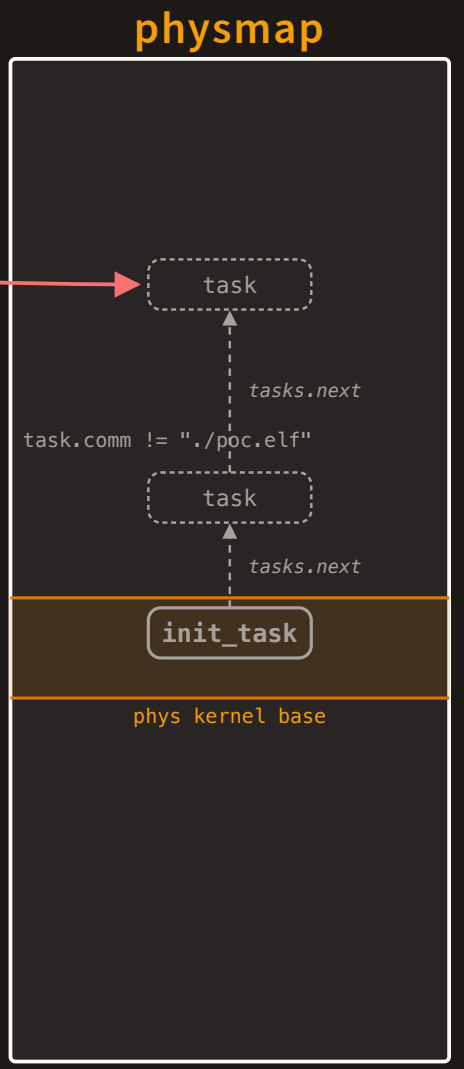
# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



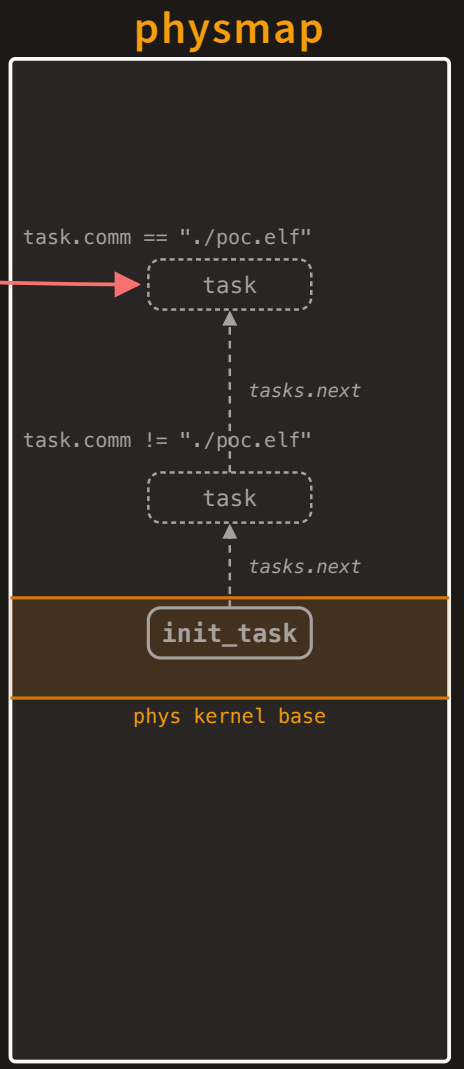
# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



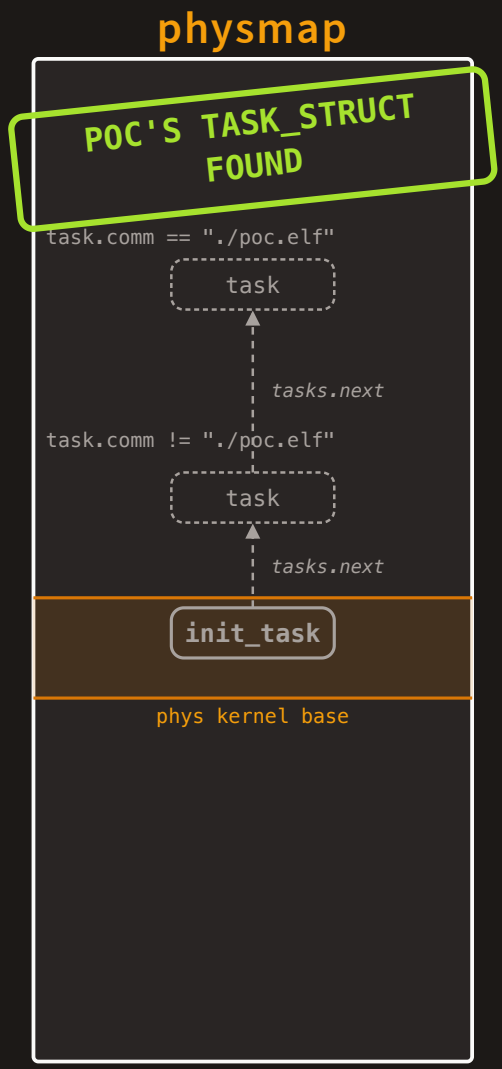
# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



# Locating `current` Task

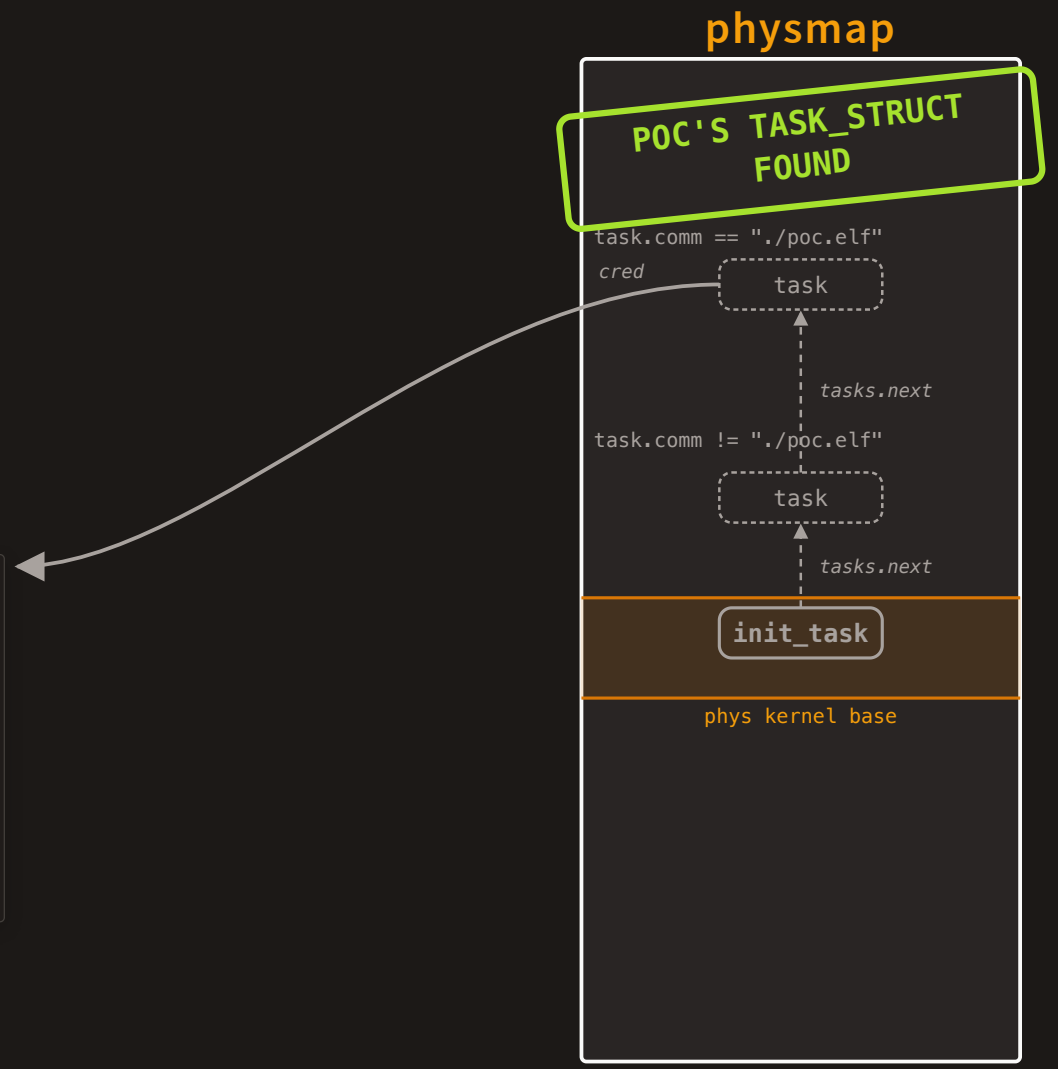
```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```



# Locating `current` Task

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

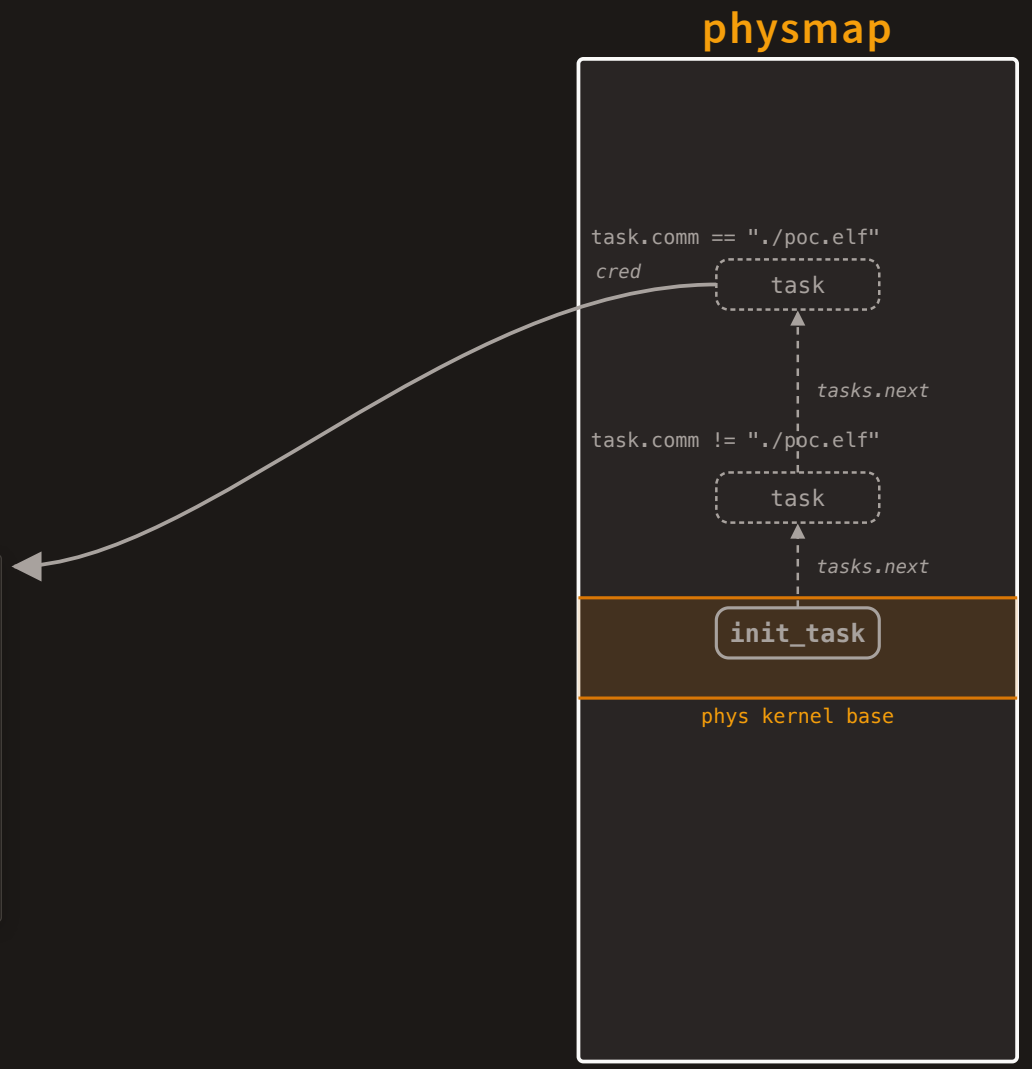
```c
struct cred {
    kuid_t uid; // real UID of the task
    kgid_t gid; // real GID of the task
    kuid_t euid; // effective UID of the task
    kgid_t egid; // effective GID of the task
    [...]
    kernel_cap_t cap_inheritable; // caps our children can inherit
    kernel_cap_t cap_permitted; // caps we're permitted
    kernel_cap_t cap_effective; // caps we can actually use
    [...]
};
```



# cred → Root

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

```c
struct cred {
    kuid_t uid; // real UID of the task
    kgid_t gid; // real GID of the task
    kuid_t euid; // effective UID of the task
    kgid_t egid; // effective GID of the task
    [...]
    kernel_cap_t cap_inheritable; // caps our children can inherit
    kernel_cap_t cap_permitted; // caps we're permitted
    kernel_cap_t cap_effective; // caps we can actually use
    [...]
};
```

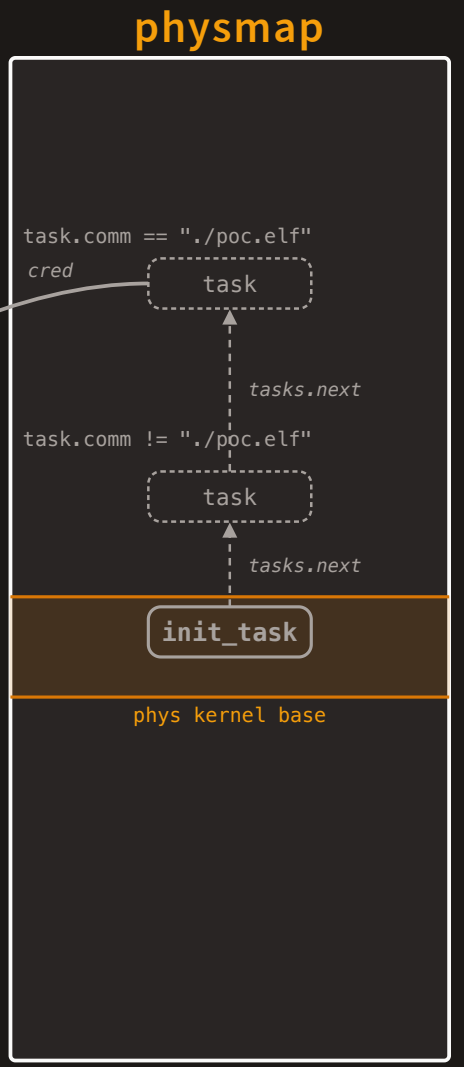


# cred → Root

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

```c
struct cred {
    kuid_t uid; // real UID of the task
    kgid_t gid; // real GID of the task
    kuid_t euid; // effective UID of the task
    kgid_t egid; // effective GID of the task
    [...]
    kernel_cap_t cap_inheritable; // caps our children can inherit
    kernel_cap_t cap_permitted; // caps we're permitted
    kernel_cap_t cap_effective; // caps we can actually use
    [...]
};
```

overwrite with 0

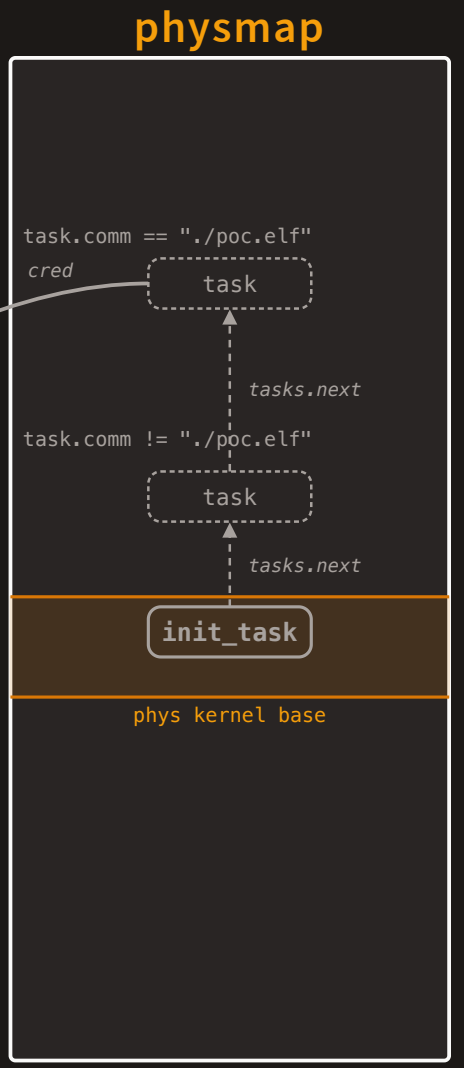


# cred → Root

```c
/* struct pipe_buffer - a linux kernel pipe buffer */
struct pipe_buffer {
    // the page containing the data for the pipe buffer
    struct page *page;
    // offset/length of data inside the @page
    unsigned int offset, len;
    // operations associated with this buffer
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

```c
struct cred {
    kuid_t uid; // real UID of the task
    kgid_t gid; // real GID of the task
    kuid_t euid; // effective UID of the task
    kgid_t egid; // effective GID of the task
    [...]
    kernel_cap_t cap_inheritable; // caps our children can inherit
    kernel_cap_t cap_permitted; // caps we're permitted
    kernel_cap_t cap_effective; // caps we can actually use
    [...]
};
```

overwrite with -1



# Testing Without Hardware

Live Demo

# Timeline

HackerOne report #3657590 · Qualcomm Product Security

# Timeline

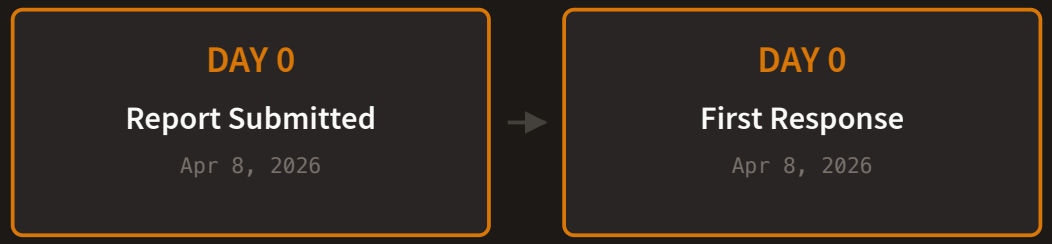
**DAY 0**

**Report Submitted**

Apr 8, 2026

HackerOne report #3657590 · Qualcomm Product Security

# Timeline




HackerOne report #3657590 · Qualcomm Product Security

# Timeline


HackerOne report #3657590 · Qualcomm Product Security

# Timeline


HackerOne report #3657590 · Qualcomm Product Security

# Timeline


HackerOne report #3657590 · Qualcomm Product Security

**A big thanks** to the Qualcomm Product Security team for the fast, professional, and transparent coordination throughout this disclosure.

# The Fix

```
static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
{
    struct qaic_bo *bo = to_qaic_bo(obj);
    unsigned long offset = 0;
    struct scatterlist *sg;

    for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {
        int ret = remap_pfn_range(vma,
                                  vma->vm_start + offset,
                                  page_to_pfn(sg_page(sg)),
                                  sg->length,
                                  vma->vm_page_prot);
        if (ret)
            return ret;
        offset += sg->length;
    }

    [...]
}
```

# The Fix

```
 static int qaic_gem_object_mmap(struct drm_gem_object *obj, struct vm_area_struct *vma)
 {
     struct qaic_bo *bo = to_qaic_bo(obj);
     unsigned long offset = 0;
+    unsigned long vma_size = vma->vm_end - vma->vm_start;
     struct scatterlist *sg;

     for (sg = bo->sgt->sgl; sg; sg = sg_next(sg)) {
+        unsigned long len = min_t(unsigned long, sg->length, vma_size - offset);
         int ret = remap_pfn_range(vma,
                                   vma->vm_start + offset,
                                   page_to_pfn(sg_page(sg)),
-                                  sg->length,
+                                  len,
                                   vma->vm_page_prot);
         if (ret)
             return ret;
-        offset += sg->length;
+        offset += len;
+        if (offset >= vma_size)
+            break;
     }

     [...]
 }
```

# Takeaways

# Takeaways

01 **Hard to mitigate.** A page UAF is an inherently hard bug class to defend against once it's present in kernel code — it operates below the abstraction layers most hardening targets.

# Takeaways

01 **Hard to mitigate.** A page UAF is an inherently hard bug class to defend against once it's present in kernel code — it operates below the abstraction layers most hardening targets.

02 **Existing hardening doesn't reach it.** Slab hardening doesn't constrain a stale mapping; the attack is data-only, so CFI doesn't apply; and no KASLR leak is needed since you read/write through a mapping you already hold.

# Takeaways

01 **Hard to mitigate.** A page UAF is an inherently hard bug class to defend against once it's present in kernel code — it operates below the abstraction layers most hardening targets.

02 **Existing hardening doesn't reach it.** Slab hardening doesn't constrain a stale mapping; the attack is data-only, so CFI doesn't apply; and no KASLR leak is needed since you read/write through a mapping you already hold.

03 **The real line of defense is access, not hardening.** Since the bug class itself is hard to close, vendors instead gate these driver nodes behind a more privileged entry point — SELinux policy, udev rules — rather than fixing the underlying primitive.

# Takeaways

01 **Hard to mitigate.** A page UAF is an inherently hard bug class to defend against once it's present in kernel code — it operates below the abstraction layers most hardening targets.

02 **Existing hardening doesn't reach it.** Slab hardening doesn't constrain a stale mapping; the attack is data-only, so CFI doesn't apply; and no KASLR leak is needed since you read/write through a mapping you already hold.

03 **The real line of defense is access, not hardening.** Since the bug class itself is hard to close, vendors instead gate these driver nodes behind a more privileged entry point — SELinux policy, udev rules — rather than fixing the underlying primitive.

04 **Not QAIC-specific.** I have found equivalent LPE primitives on other Qualcomm-based and OEM Android devices — Samsung, Xiaomi, Oppo, Huawei.

Calif Research

# Privilege Escalation via a Page Use-After-Free in Qualcomm's QAIC Driver

SPEAKER
Lukas Maar

CONFERENCE
PRASEC

DATE
September 10, 2026