# DPDK HASH LIB

Saikrishna Edupuganti
Intel Labs

krsna1729/dpdk-hash

# Sections

- Hash and Hash-Table
  - 101
- Skeleton App
  - Extend this app to filter packets by using the Hash library
- DPDK RTE_HASH
  - Library internals
- Test App
  - Unit test and perf tests. Look at hash related tests
- Benchmark changes to lib
  - Modify Add procedure. Use test app to check the effect

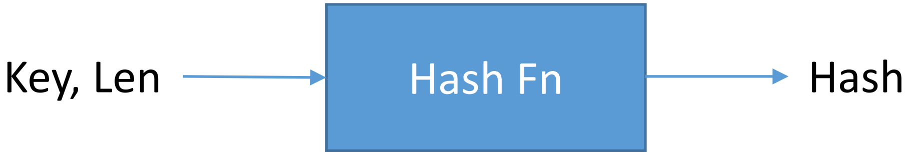
# Sections

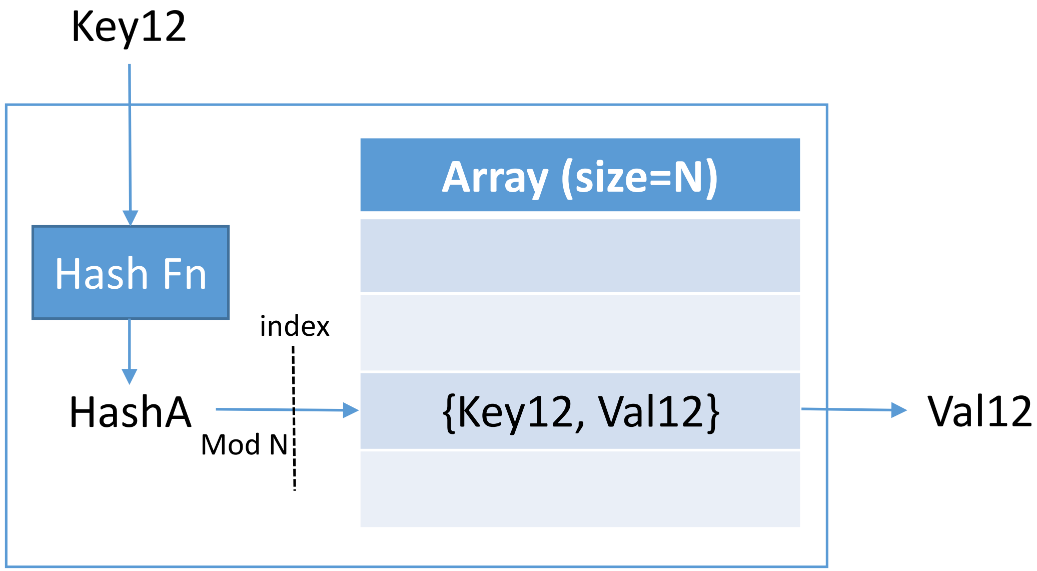
- Hash and Hash-Table
  - 101
- Skeleton App
  - Extend this app to filter packets by using the Hash library
- DPDK RTE_HASH
  - Library internals
- Test App
  - Unit test and perf tests. Look at hash related tests
- Benchmark changes to lib
  - Modify Add procedure. Use test app to check the effect

# Hash and Hash Table

- Hash fn–
  - Jhash
  - Murmur
  - CRC

- Hash table –
  - Google {Sparse, dense}_hash_map
  - GCC unordered_map
  - Python dict





(Hash table with fixed length key/val pictured here)

http://incise.org/hash-table-benchmarks.html

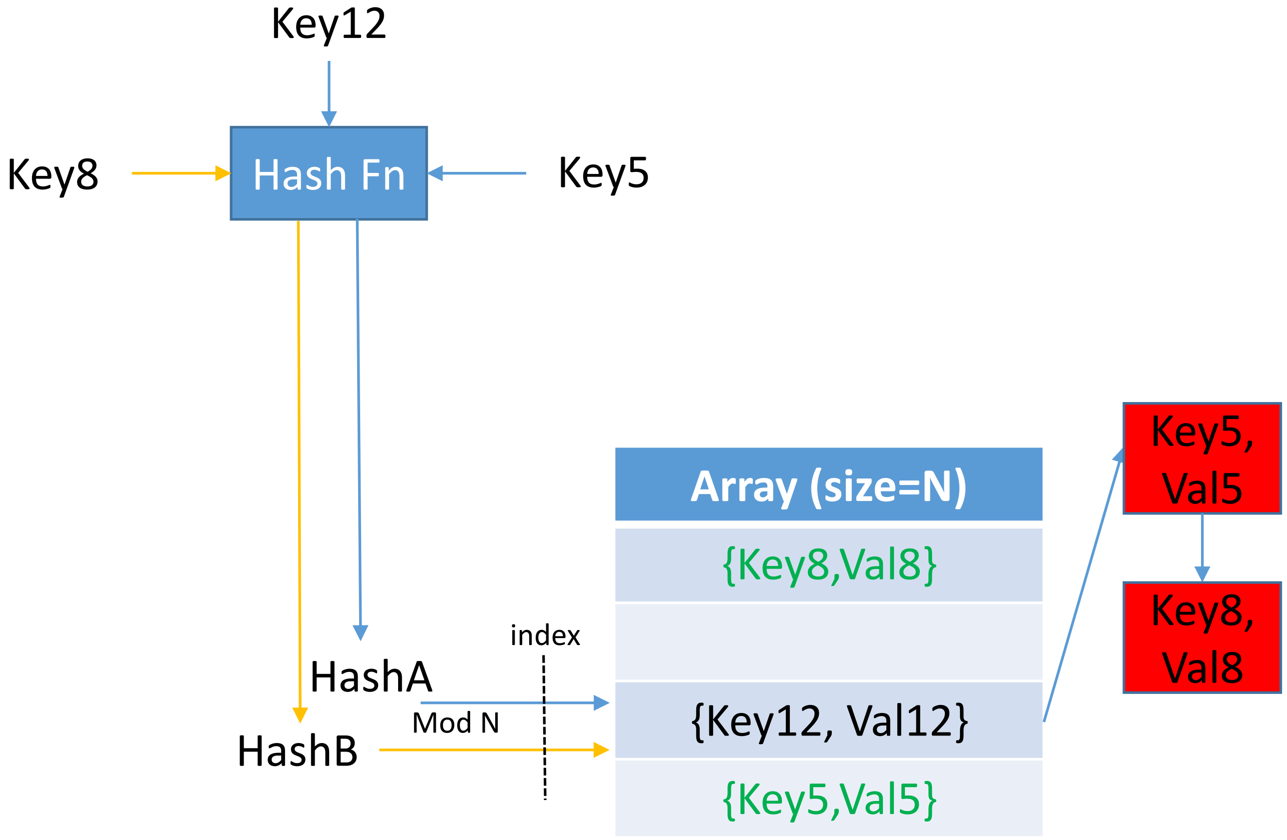
# Hash Table - Collision resolution

Affects Util/Add/Lookup/Del Perf

- Chaining:
  - Linked list
  - List head cells
  - Other structures

- Open addressing:
  - Probe – Linear, quadratic, double
  - Move – Cuckoo (today's focus)

https://en.wikipedia.org/wiki/Hash_table

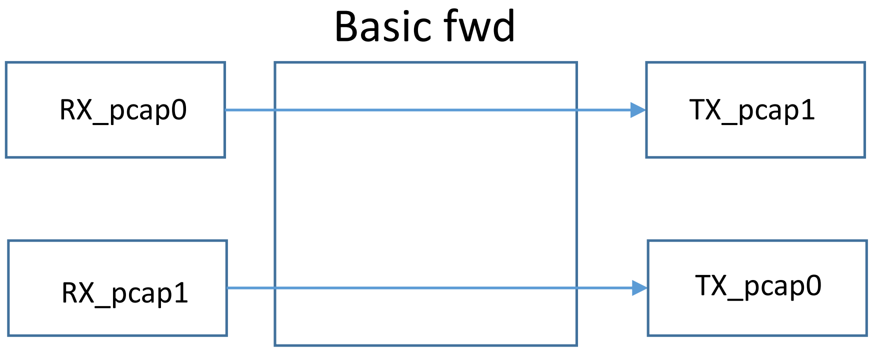


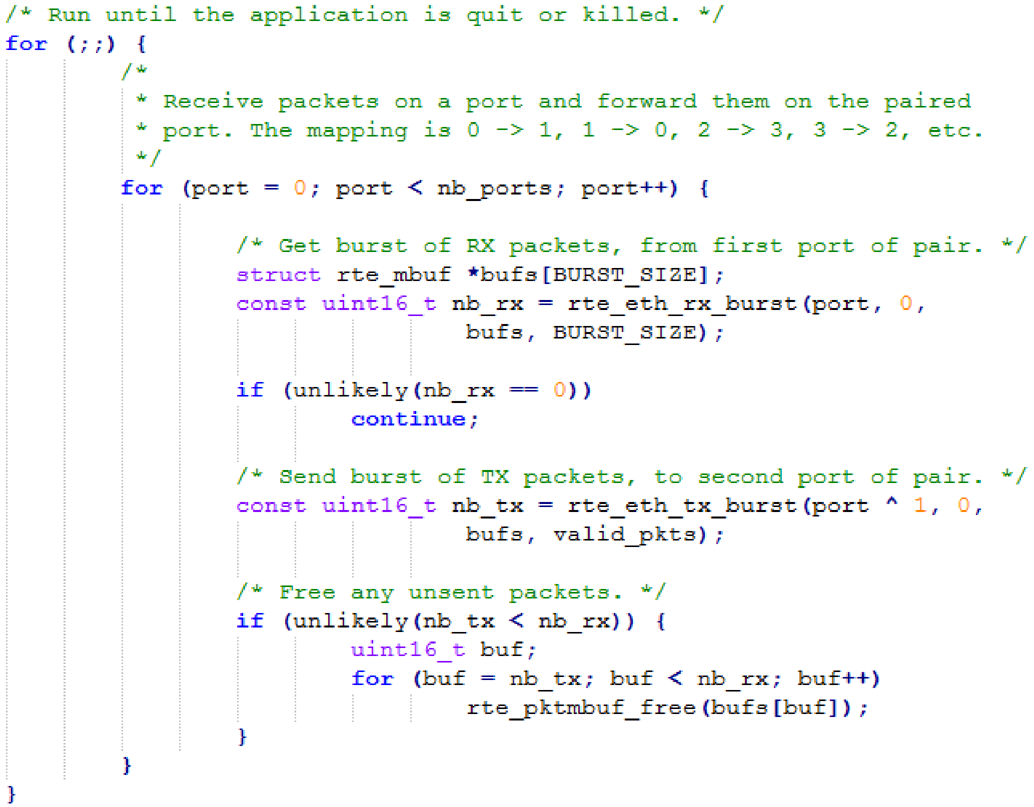
# Sections

- Hash and Hash-Table
  - 101
- Skeleton App
  - Extend this app to filter packets by using the Hash library
- DPDK RTE_HASH
  - Library internals
- Test App
  - Unit test and perf tests. Look at hash related tests
- Benchmark changes to lib
  - Modify Add procedure. Use test app to check the effect

# Skeleton App

- Basic forwarding
  - Port 0 -> Port 1
  - Port 1 -> Port 0
  - Output PCAPs same size as Input PCAPs





```
/* Run until the application is quit or killed. */
for (;;) {
        /*
         * Receive packets on a port and forward them on the paired
         * port. The mapping is 0 -> 1, 1 -> 0, 2 -> 3, 3 -> 2, etc.
         */
        for (port = 0; port < nb_ports; port++) {

                /* Get burst of RX packets, from first port of pair. */
                struct rte_mbuf *bufs[BURST_SIZE];
                const uint16_t nb_rx = rte_eth_rx_burst(port, 0,
                                bufs, BURST_SIZE);

                if (unlikely(nb_rx == 0))
                        continue;

                /* Send burst of TX packets, to second port of pair. */
                const uint16_t nb_tx = rte_eth_tx_burst(port ^ 1, 0,
                                bufs, valid_pkts);

                /* Free any unsent packets. */
                if (unlikely(nb_tx < nb_rx)) {
                        uint16_t buf;
                        for (buf = nb_tx; buf < nb_rx; buf++)
                                rte_pktmbuf_free(bufs[buf]);
                }
        }
}
```

# Skeleton App

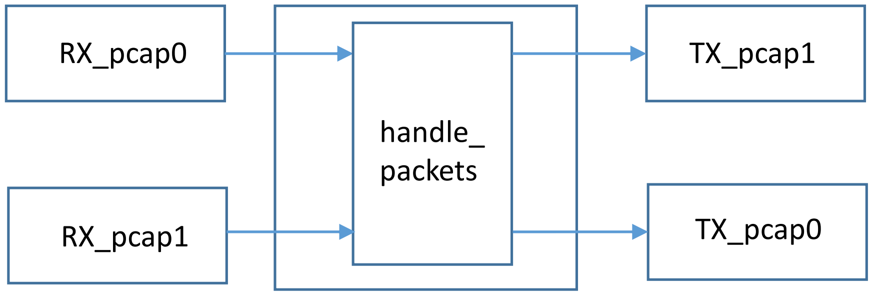
- Basic forwarding

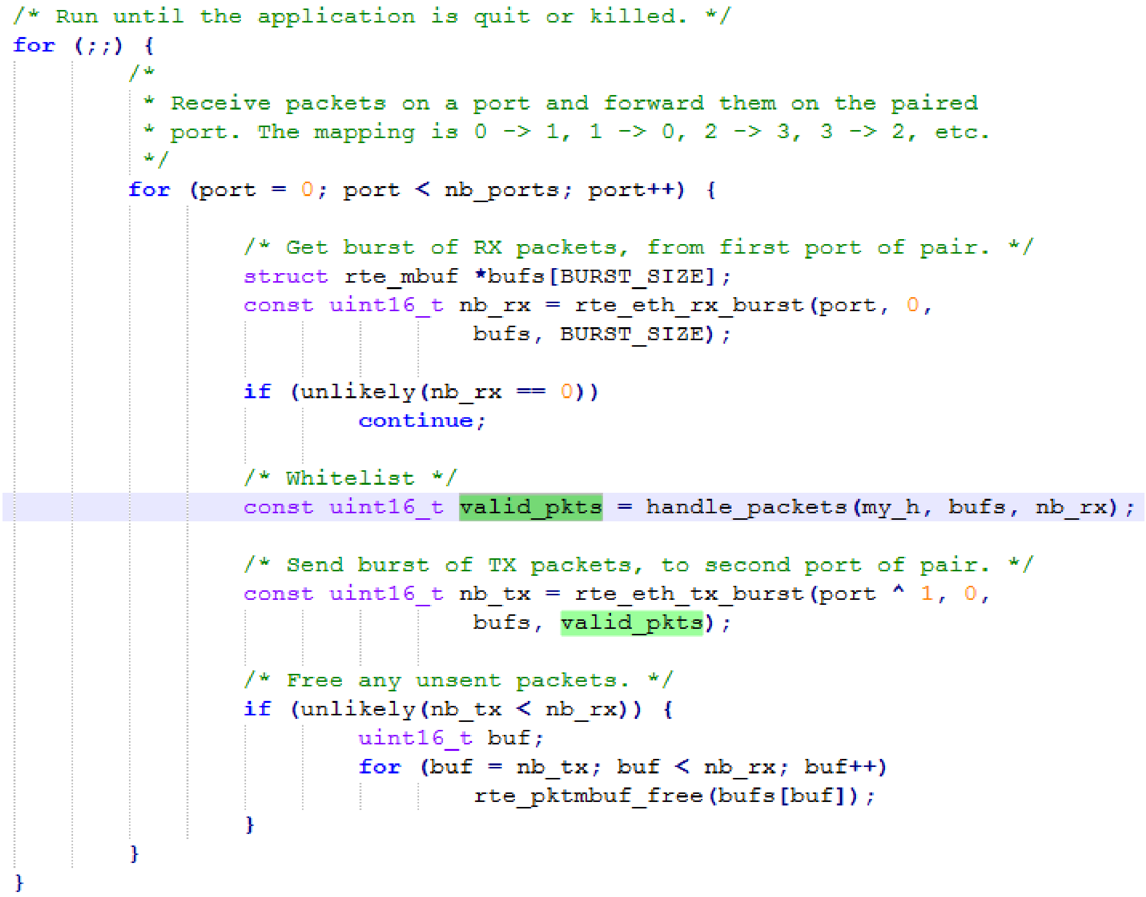
```
ll *.pcap    # should see no PCAP files
make -C examples/skeleton/
sudo ./examples/skeleton/build/basicfwd --no-pci \
--vdev=net_pcap0,rx_pcap=/vagrant/pcaps/64K_dst0.pcap,tx_pcap=b_output0.pcap \
--vdev=net_pcap1,rx_pcap=/vagrant/pcaps/64K_dst1.pcap,tx_pcap=b_output1.pcap
# Press Ctrl+C to quit after 2s
ll *.pcap   # should see 2 new PCAP files b_output0 b_output1
```

# Skeleton App

- Lets make it a bit more intelligent
  - Create a hash-table of permitted Dst IPs
  - Insert rules for half IPs
  - Drop packets if miss
  - Output PCAPs size should reduce to half

Whitelist fwd





```
/* Run until the application is quit or killed. */
for (;;) {
    /*
     * Receive packets on a port and forward them on the paired
     * port. The mapping is 0 -> 1, 1 -> 0, 2 -> 3, 3 -> 2, etc.
     */
    for (port = 0; port < nb_ports; port++) {

        /* Get burst of RX packets, from first port of pair. */
        struct rte_mbuf *bufs[BURST_SIZE];
        const uint16_t nb_rx = rte_eth_rx_burst(port, 0,
                bufs, BURST_SIZE);

        if (unlikely(nb_rx == 0))
            continue;

        /* Whitelist */
        const uint16_t valid_pkts = handle_packets(my_h, bufs, nb_rx);

        /* Send burst of TX packets, to second port of pair. */
        const uint16_t nb_tx = rte_eth_tx_burst(port ^ 1, 0,
                bufs, valid_pkts);

        /* Free any unsent packets. */
        if (unlikely(nb_tx < nb_rx)) {
            uint16_t buf;
            for (buf = nb_tx; buf < nb_rx; buf++)
                rte_pktmbuf_free(bufs[buf]);
        }
    }
}
```

# Skeleton App

- Lets make it a bit more intelligent
  - Create   rte_hash_create (const struct rte_hash_parameters *params)
  - Add      rte_hash_add_key (const struct rte_hash *h, const void *key)
  - Lookup  rte_hash_lookup_bulk (const struct rte_hash *h, const void **keys, uint32_t num_keys, int32_t *positions)

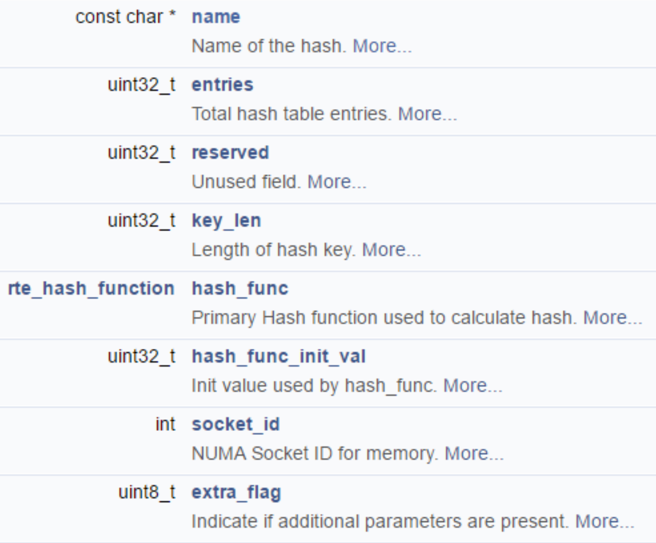
Delete (assignment)

http://dpdk.org/doc/api/rte__hash_8h.html

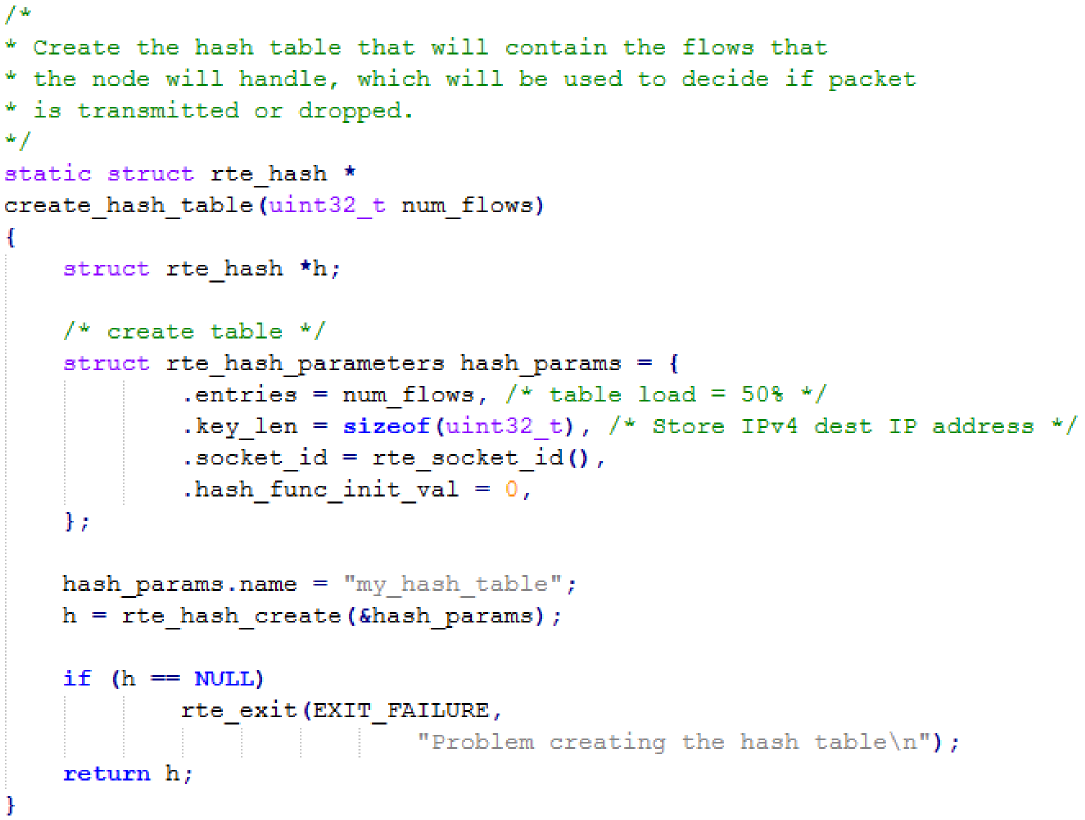
# Skeleton App

- Lets make it a bit more intelligent
  - Create

rte_hash_create (
const struct rte_hash_parameters *params)



| | | |
|---|---|---|
| const char * | **name** | Name of the hash. More... |
| uint32_t | **entries** | Total hash table entries. More... |
| uint32_t | **reserved** | Unused field. More... |
| uint32_t | **key_len** | Length of hash key. More... |
| **rte_hash_function** | **hash_func** | Primary Hash function used to calculate hash. More... |
| uint32_t | **hash_func_init_val** | Init value used by hash_func. More... |
| int | **socket_id** | NUMA Socket ID for memory. More... |
| uint8_t | **extra_flag** | Indicate if additional parameters are present. More... |



```
/*
* Create the hash table that will contain the flows that
* the node will handle, which will be used to decide if packet
* is transmitted or dropped.
*/
static struct rte_hash *
create_hash_table(uint32_t num_flows)
{
    struct rte_hash *h;

    /* create table */
    struct rte_hash_parameters hash_params = {
            .entries = num_flows, /* table load = 50% */
            .key_len = sizeof(uint32_t), /* Store IPv4 dest IP address */
            .socket_id = rte_socket_id(),
            .hash_func_init_val = 0,
    };

    hash_params.name = "my_hash_table";
    h = rte_hash_create(&hash_params);

    if (h == NULL)
            rte_exit(EXIT_FAILURE,
                                    "Problem creating the hash table\n");
    return h;
}
```

num_flows = 1<<16
unique Dst IP in PCAPs = 1<<16

http://dpdk.org/doc/api/rte__hash_8h.html#a5afbd2564f738149a241bc22b2428612
http://dpdk.org/doc/api/structrte__hash__parameters.html#a8f8f80d37794cde9472343e4487ba3eb

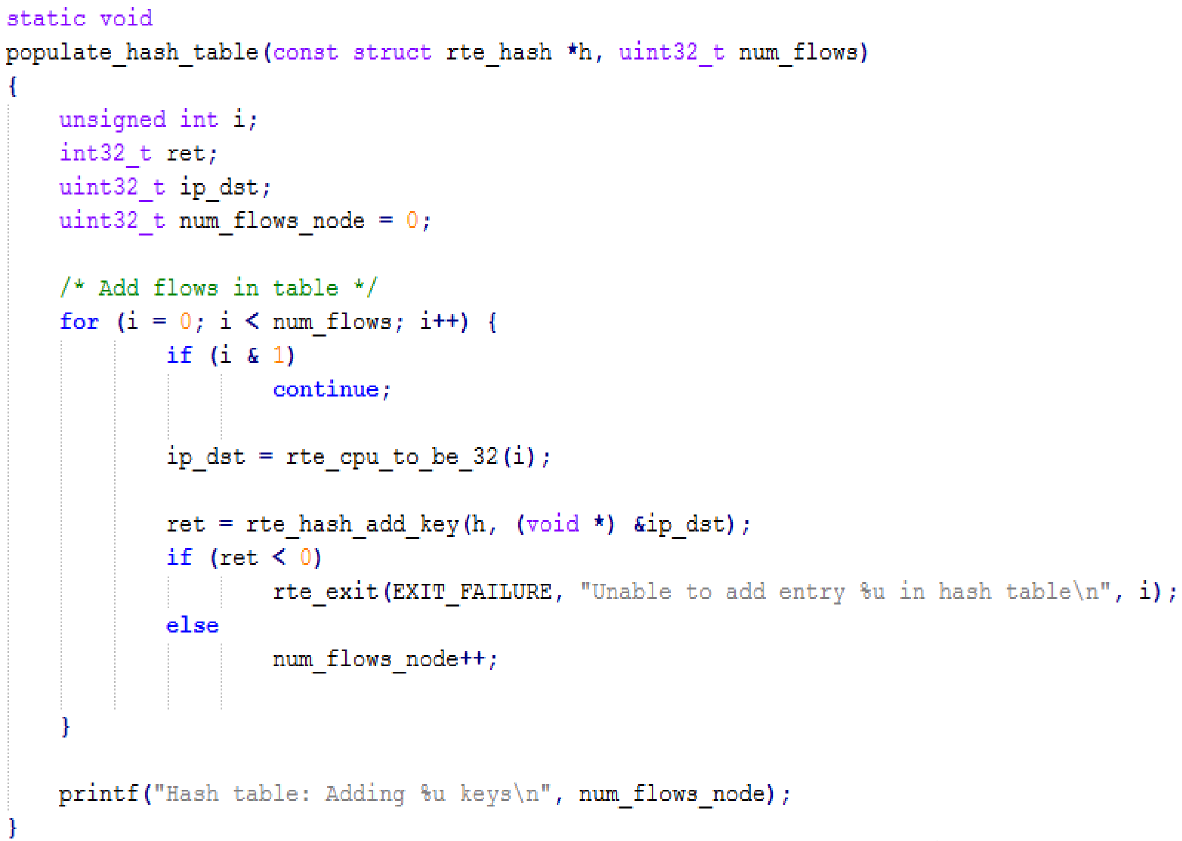
# Skeleton App

- Lets make it a bit more intelligent
    - Add
    - Insert even Dst Ips
    - No value stored in this usage
    - (could have had array of actions, uses ret as index)

rte_hash_add_key (
const struct rte_hash *h, const void *key)
rte_hash_add_key_data

rte_hash_add_key_with_hash
rte_hash_add_key_with_hash_data

[http://dpdk.org/doc/api/rte__hash_8h.html#a0247f58baa6cbf614e5b729ff0baf27e](http://dpdk.org/doc/api/rte__hash_8h.html#a0247f58baa6cbf614e5b729ff0baf27e)



```
static void
populate_hash_table(const struct rte_hash *h, uint32_t num_flows)
{
    unsigned int i;
    int32_t ret;
    uint32_t ip_dst;
    uint32_t num_flows_node = 0;

    /* Add flows in table */
    for (i = 0; i < num_flows; i++) {
        if (i & 1)
            continue;

        ip_dst = rte_cpu_to_be_32(i);

        ret = rte_hash_add_key(h, (void *) &ip_dst);
        if (ret < 0)
            rte_exit(EXIT_FAILURE, "Unable to add entry %u in hash table\n", i);
        else
            num_flows_node++;

    }

    printf("Hash table: Adding %u keys\n", num_flows_node);
}
```

rules added = (1<<16)/2

# Skeleton App

- Lets make it a bit more intelligent
  - Lookup

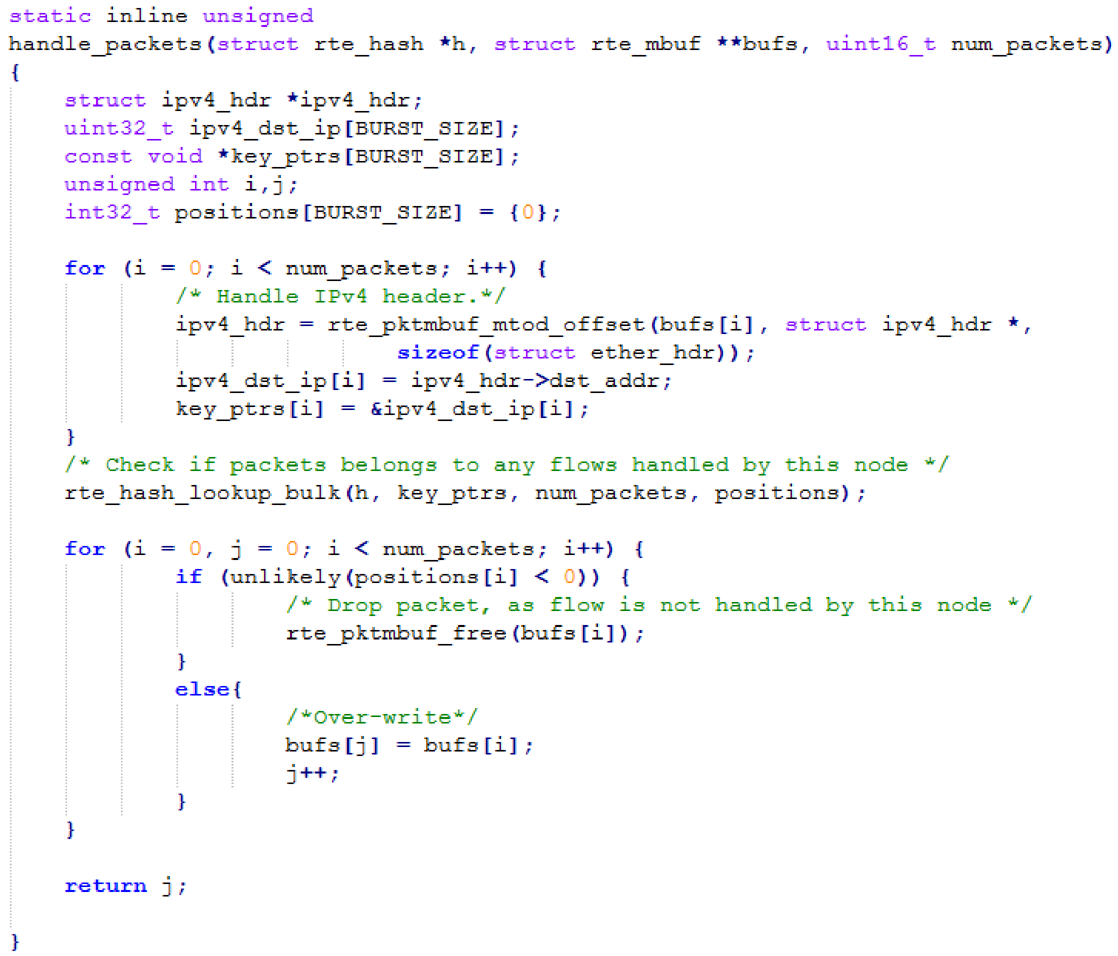
rte_hash_lookup

rte_hash_lookup_bulk (
const struct rte_hash *h, const void **keys,
uint32_t num_keys, int32_t *positions)

rte_hash_lookup_data
rte_hash_lookup_bulk_data

rte_hash_lookup_with_hash
rte_hash_lookup_with_hash_data

[http://dpdk.org/doc/api/rte__hash_8h.html#a420dedbd249c73bbb94a98e10a87b088](http://dpdk.org/doc/api/rte__hash_8h.html#a420dedbd249c73bbb94a98e10a87b088)



```
static inline unsigned
handle_packets(struct rte_hash *h, struct rte_mbuf **bufs, uint16_t num_packets)
{
    struct ipv4_hdr *ipv4_hdr;
    uint32_t ipv4_dst_ip[BURST_SIZE];
    const void *key_ptrs[BURST_SIZE];
    unsigned int i,j;
    int32_t positions[BURST_SIZE] = {0};

    for (i = 0; i < num_packets; i++) {
        /* Handle IPv4 header.*/
        ipv4_hdr = rte_pktmbuf_mtod_offset(bufs[i], struct ipv4_hdr *,
                sizeof(struct ether_hdr));
        ipv4_dst_ip[i] = ipv4_hdr->dst_addr;
        key_ptrs[i] = &ipv4_dst_ip[i];
    }
    /* Check if packets belongs to any flows handled by this node */
    rte_hash_lookup_bulk(h, key_ptrs, num_packets, positions);

    for (i = 0, j = 0; i < num_packets; i++) {
        if (unlikely(positions[i] < 0)) {
            /* Drop packet, as flow is not handled by this node */
            rte_pktmbuf_free(bufs[i]);
        }
        else{
            /*Over-write*/
            bufs[j] = bufs[i];
            j++;
        }
    }

    return j;

}
```

# Skeleton App

- Lets make it a bit more intelligent
  - Delete
  - Try to delete half the rules added immediately after Add.
  - Output PCAP should be 1/4th
  - Assignment for you ☺

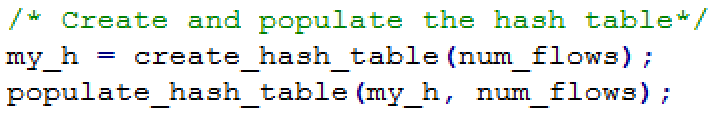
rte_hash_del_key (const struct rte_hash *h, const void *key)

rte_hash_del_key_with_hash

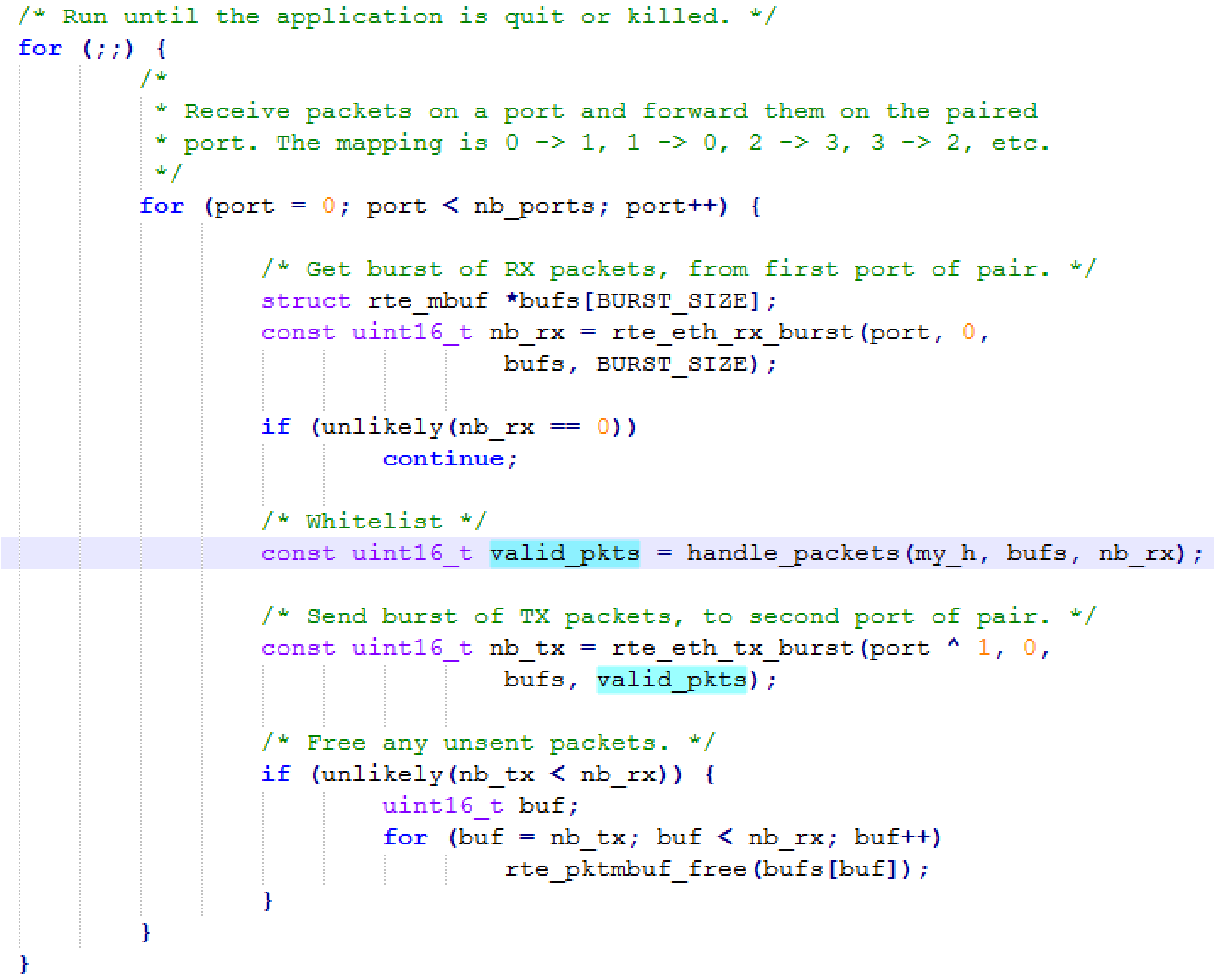
http://dpdk.org/doc/api/rte__hash_8h.html#aaa125f9c9f57ea5fdf2752bcb506a058

# Skeleton App

- Finally it’s a bit more intelligent



```
/* Create and populate the hash table*/
my_h = create_hash_table(num_flows);
populate_hash_table(my_h, num_flows);
```



```
/* Run until the application is quit or killed. */
for (;;) {
        /*
         * Receive packets on a port and forward them on the paired
         * port. The mapping is 0 -> 1, 1 -> 0, 2 -> 3, 3 -> 2, etc.
         */
        for (port = 0; port < nb_ports; port++) {

                /* Get burst of RX packets, from first port of pair. */
                struct rte_mbuf *bufs[BURST_SIZE];
                const uint16_t nb_rx = rte_eth_rx_burst(port, 0,
                                bufs, BURST_SIZE);

                if (unlikely(nb_rx == 0))
                        continue;

                /* Whitelist */
                const uint16_t valid_pkts = handle_packets(my_h, bufs, nb_rx);

                /* Send burst of TX packets, to second port of pair. */
                const uint16_t nb_tx = rte_eth_tx_burst(port ^ 1, 0,
                                bufs, valid_pkts);

                /* Free any unsent packets. */
                if (unlikely(nb_tx < nb_rx)) {
                        uint16_t buf;
                        for (buf = nb_tx; buf < nb_rx; buf++)
                                rte_pktmbuf_free(bufs[buf]);
                }
        }
}
```

[http://dpdk.org/doc/api/rte__hash_8h.html](http://dpdk.org/doc/api/rte__hash_8h.html)

# Skeleton App

- Finally it’s a bit more intelligent

```
git apply /vagrant/patches/skeleton_whitelist_ips.patch
ll *.pcap
make -C examples/skeleton/
sudo ./examples/skeleton/build/basicfwd --no-pci \
--vdev=net_pcap0,rx_pcap=/vagrant/pcaps/64K_dst0.pcap,tx_pcap=a_output0.pcap \
--vdev=net_pcap1,rx_pcap=/vagrant/pcaps/64K_dst1.pcap,tx_pcap=a_output1.pcap
# Press Ctrl+C to quit after 2s
ll *.pcap   # should see 2 new PCAP files a_output0 a_output1 half the size
```

# Sections

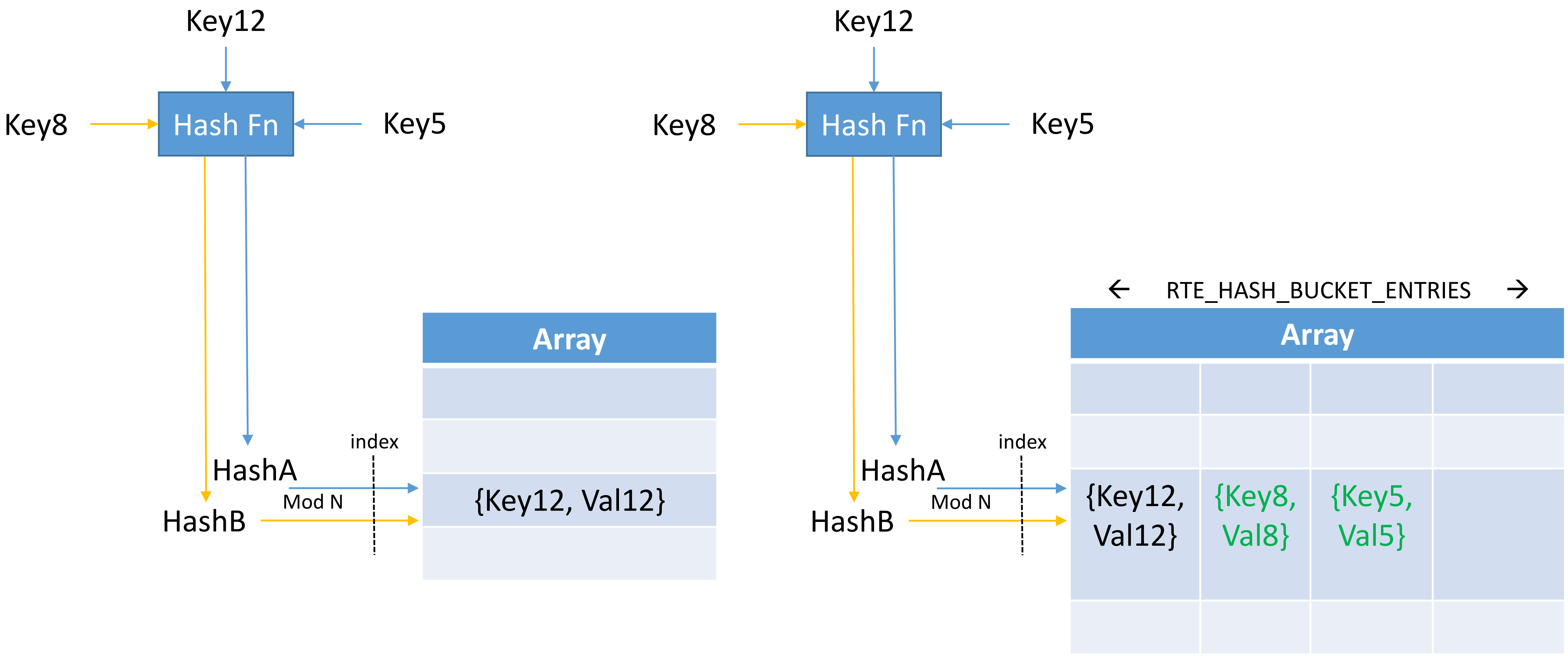
- Hash and Hash-Table
    - 101
- Skeleton App
    - Extend this app to filter packets by using the Hash library
- DPDK RTE_HASH
    - Library internals
- Test App
    - Unit test and perf tests. Look at hash related tests
- Benchmark changes to lib
    - Modify Add procedure. Use test app to check the effect

# DPDK RTE_HASH

Internals

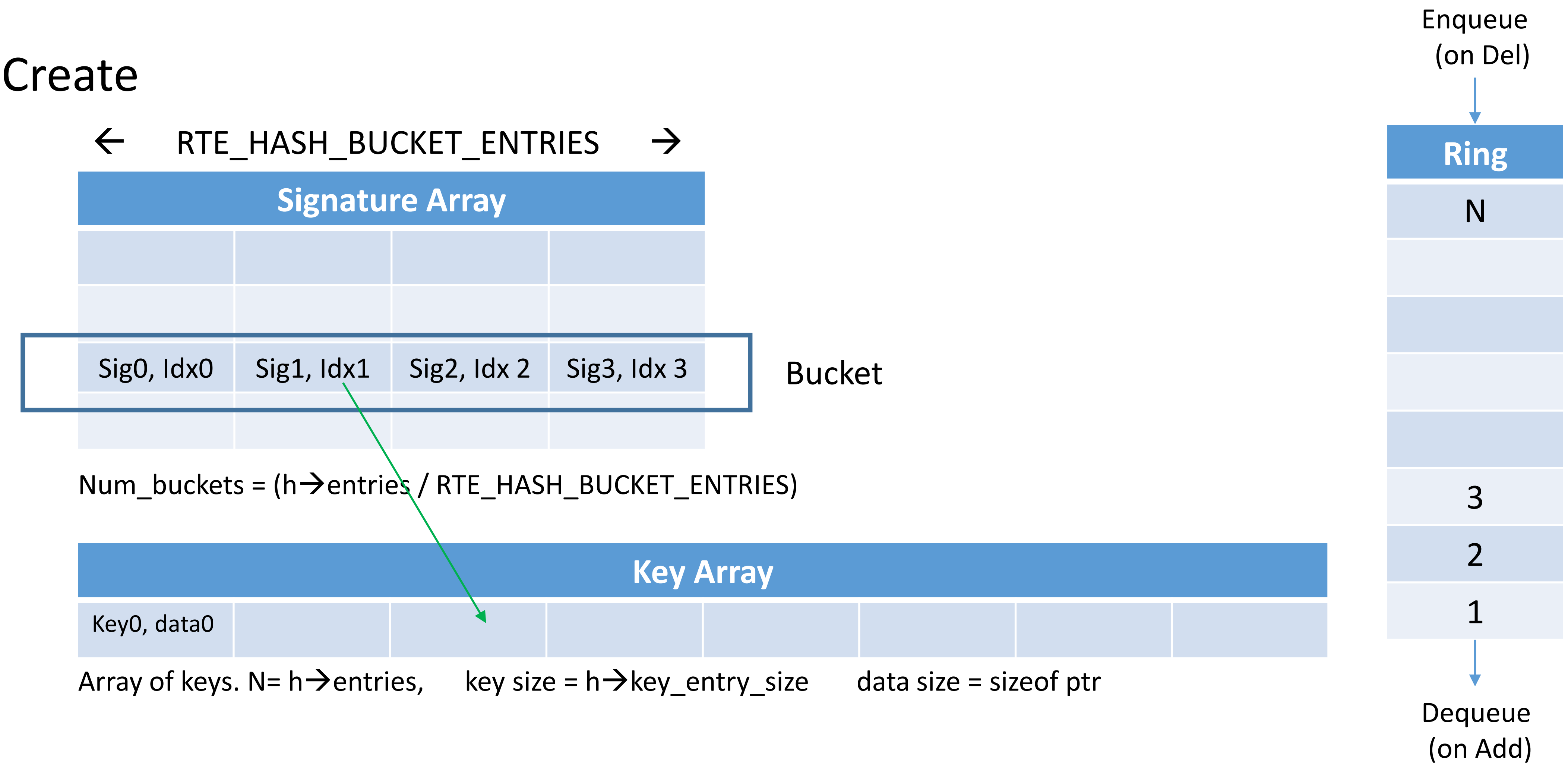
- Create
- Lookup
- Delete
- Add

# DPDK RTE_HASH



# DPDK RTE_HASH

- Create



← RTE_HASH_BUCKET_ENTRIES →

**Signature Array**

Sig0, Idx0 | Sig1, Idx1 | Sig2, Idx 2 | Sig3, Idx 3

Bucket

Num_buckets = (h→entries / RTE_HASH_BUCKET_ENTRIES)

**Key Array**

Key0, data0

Array of keys. N= h→entries, key size = h→key_entry_size data size = sizeof ptr

Enqueue
(on Del)

**Ring**

N

3

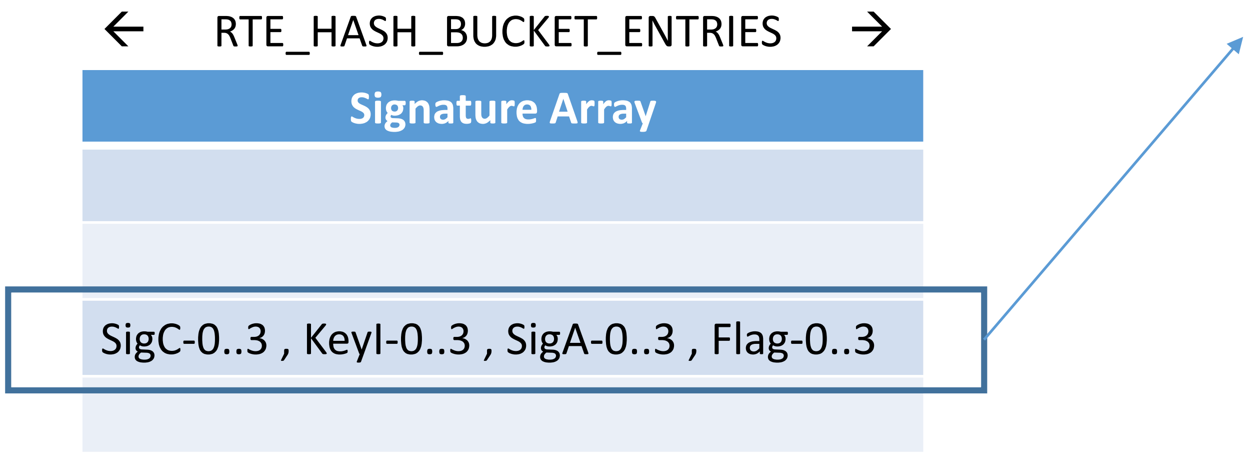
2

1

Dequeue
(on Add)

# DPDK RTE_HASH

- Create

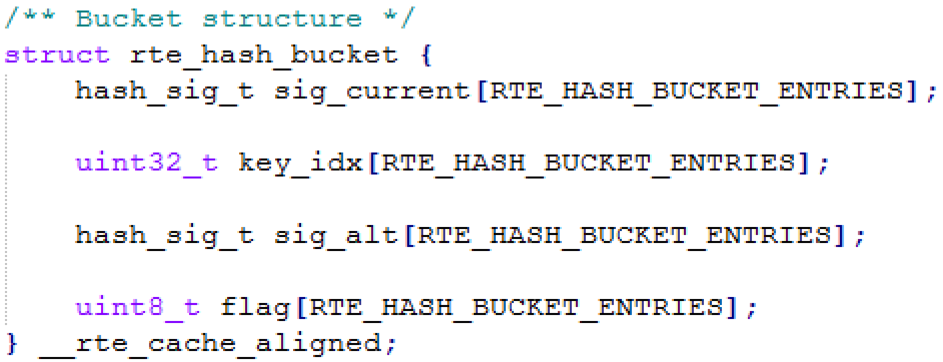


← RTE_HASH_BUCKET_ENTRIES →

**Signature Array**

SigC-0..3 , KeyI-0..3 , SigA-0..3 , Flag-0..3

N= (h→entries / RTE_HASH_BUCKET_ENTRIES)

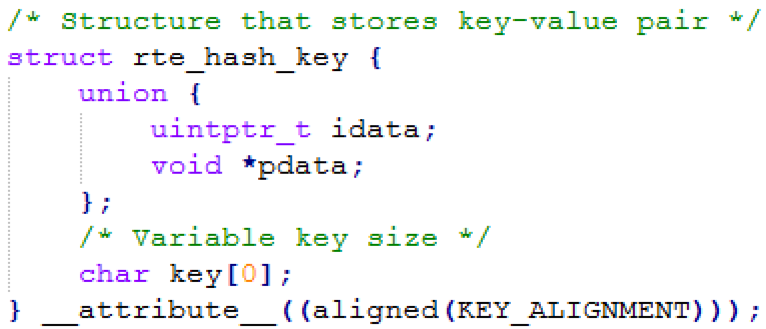


```
/** Bucket structure */
struct rte_hash_bucket {
    hash_sig_t sig_current[RTE_HASH_BUCKET_ENTRIES];

    uint32_t key_idx[RTE_HASH_BUCKET_ENTRIES];

    hash_sig_t sig_alt[RTE_HASH_BUCKET_ENTRIES];

    uint8_t flag[RTE_HASH_BUCKET_ENTRIES];
} __rte_cache_aligned;
```

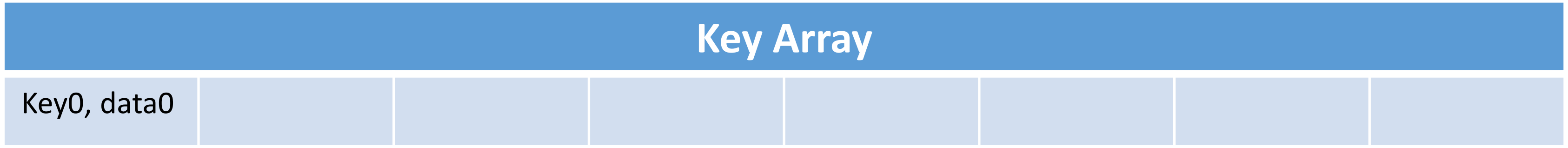
More like this.
Layout helps do vectorized lookups → AoS, SoA, AoSoA
Sig_cur and key_idx → first cache-line → needed for Lookups
Sig_alt and flag → next cache-line → needed for Adds

# DPDK RTE_HASH

- Create



```
/* Structure that stores key-value pair */
struct rte_hash_key {
    union {
        uintptr_t idata;
        void *pdata;
    };
    /* Variable key size */
    char key[0];
} __attribute__((aligned(KEY_ALIGNMENT)));
```



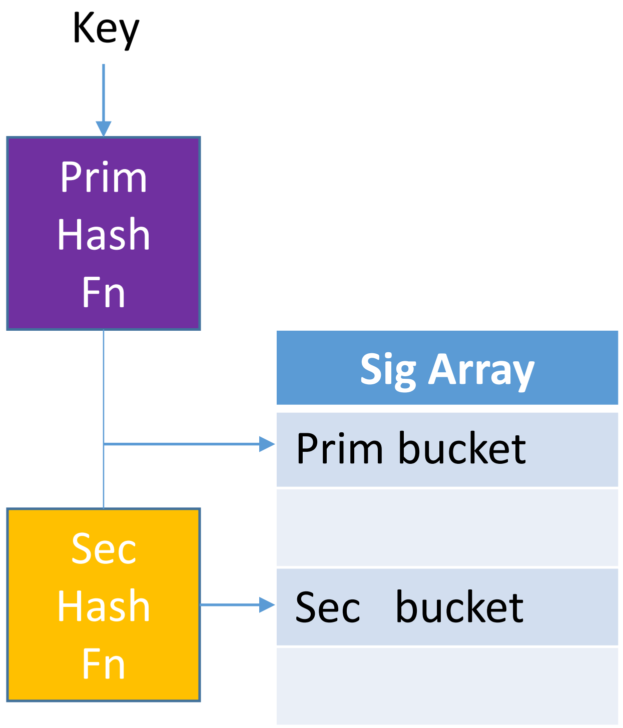
**Key Array**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Key0, data0 | | | | | | | |

Array of keys. N= h→entries, key size = h→key_entry_size data size = sizeof ptr

# DPDK RTE_HASH

- Lookup

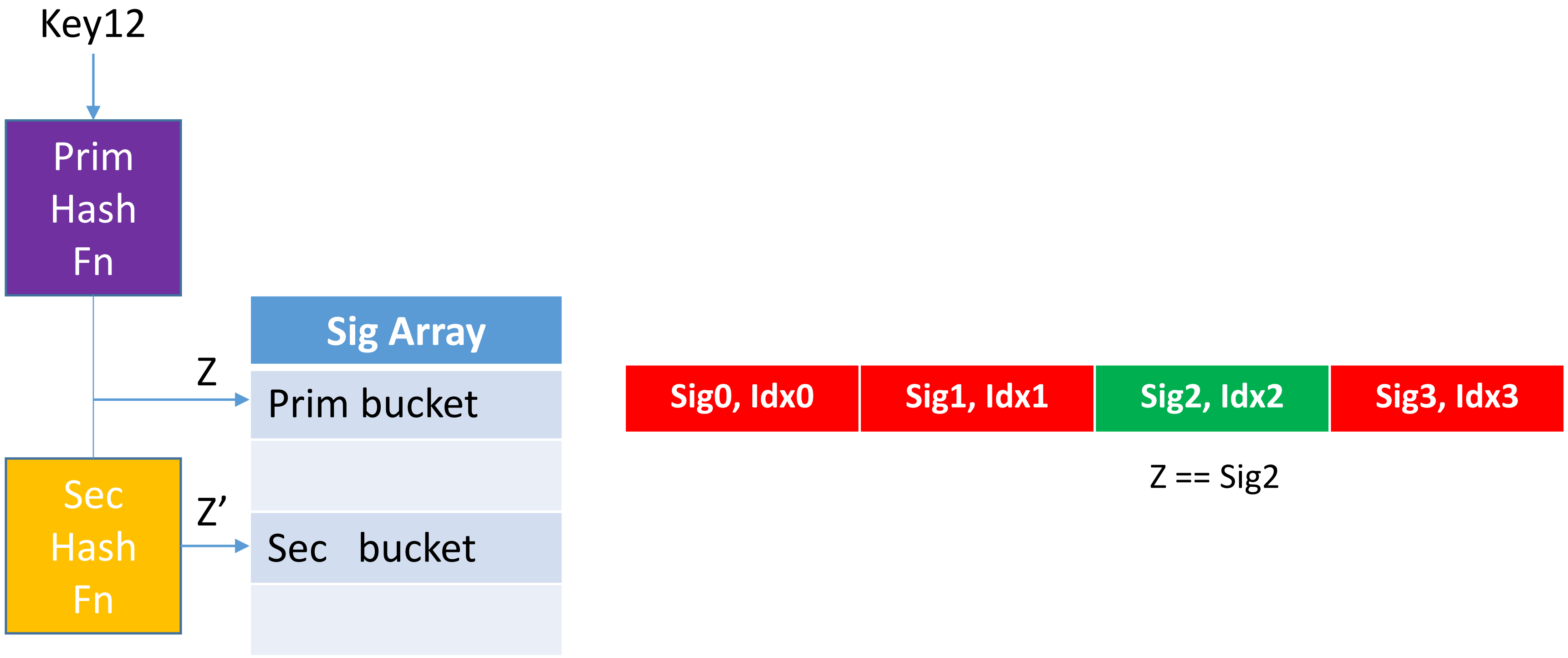


We look at the only 2 locations this key can be.

```
bucket_idx = sig & h->bucket_bitmask;
bkt = &h->buckets[bucket_idx];

/* Check if key is in primary location */
for (i = 0; i < RTE_HASH_BUCKET_ENTRIES; i++) {
    if (bkt->sig_current[i] == sig &&
            bkt->key_idx[i] != EMPTY_SLOT) {

            /* ptr math to access the stored key k*/

            /* Check if keys are match*/
            if (rte_hash_cmp_eq(key, k->key, h) == 0) {
                /*
                 * Return index where key is stored,
                 * substracting the first dummy index
                 */
                return bkt->key_idx[i] - 1;

/* Rinse and repeat above*/

/* Calculate secondary hash */
alt_hash = rte_hash_secondary_hash(sig);
bucket_idx = alt_hash & h->bucket_bitmask;
bkt = &h->buckets[bucket_idx];

/* Check if key is in seconday location */
```

# DPDK RTE_HASH

- Lookup



Key12

Prim Hash Fn

Z

Z'

Sec Hash Fn

Sig Array

Prim bucket

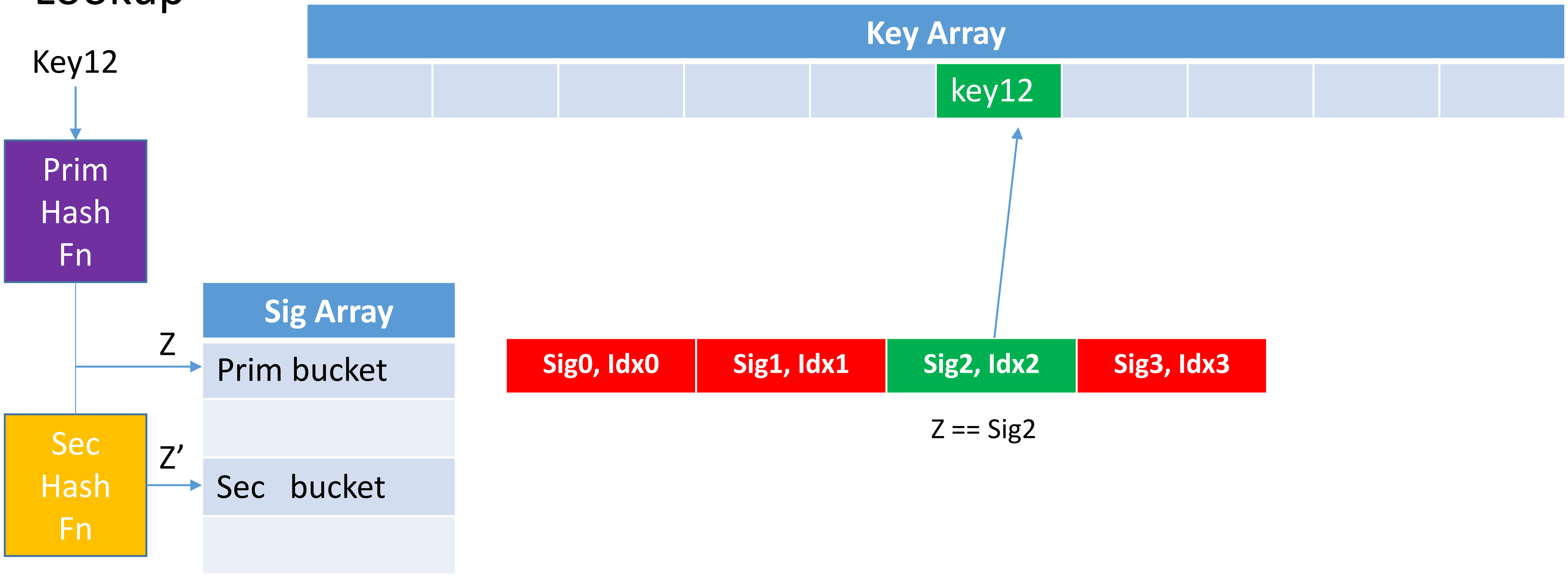
Sec bucket

| Sig0, Idx0 | Sig1, Idx1 | Sig2, Idx2 | Sig3, Idx3 |
|---|---|---|---|

Z == Sig2

# DPDK RTE_HASH

- Lookup

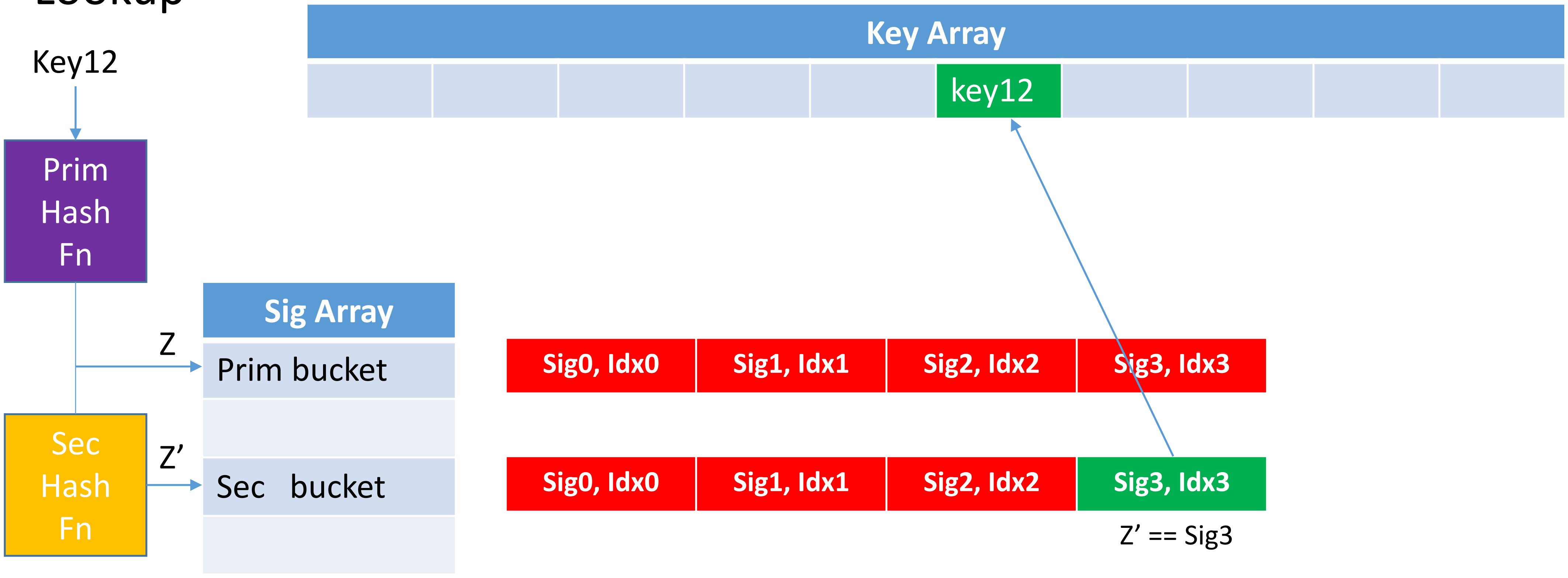


Z == Sig2

Return the index on match

# DPDK RTE_HASH

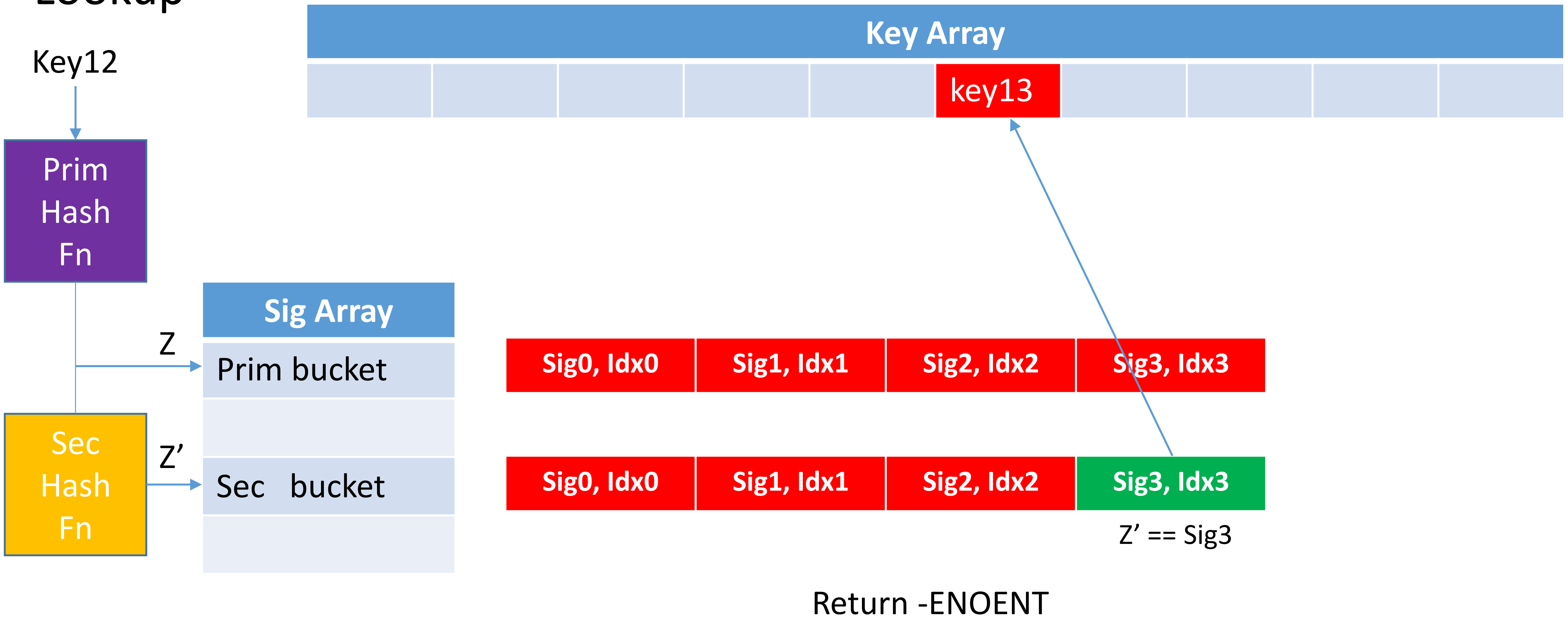
- Lookup



Return the index on match

# DPDK RTE_HASH

- Lookup



Key12

Prim Hash Fn

Sec Hash Fn

Z

Z’

**Sig Array**

Prim bucket

Sec bucket

**Key Array**

key13

**Sig0, Idx0** | **Sig1, Idx1** | **Sig2, Idx2** | **Sig3, Idx3**

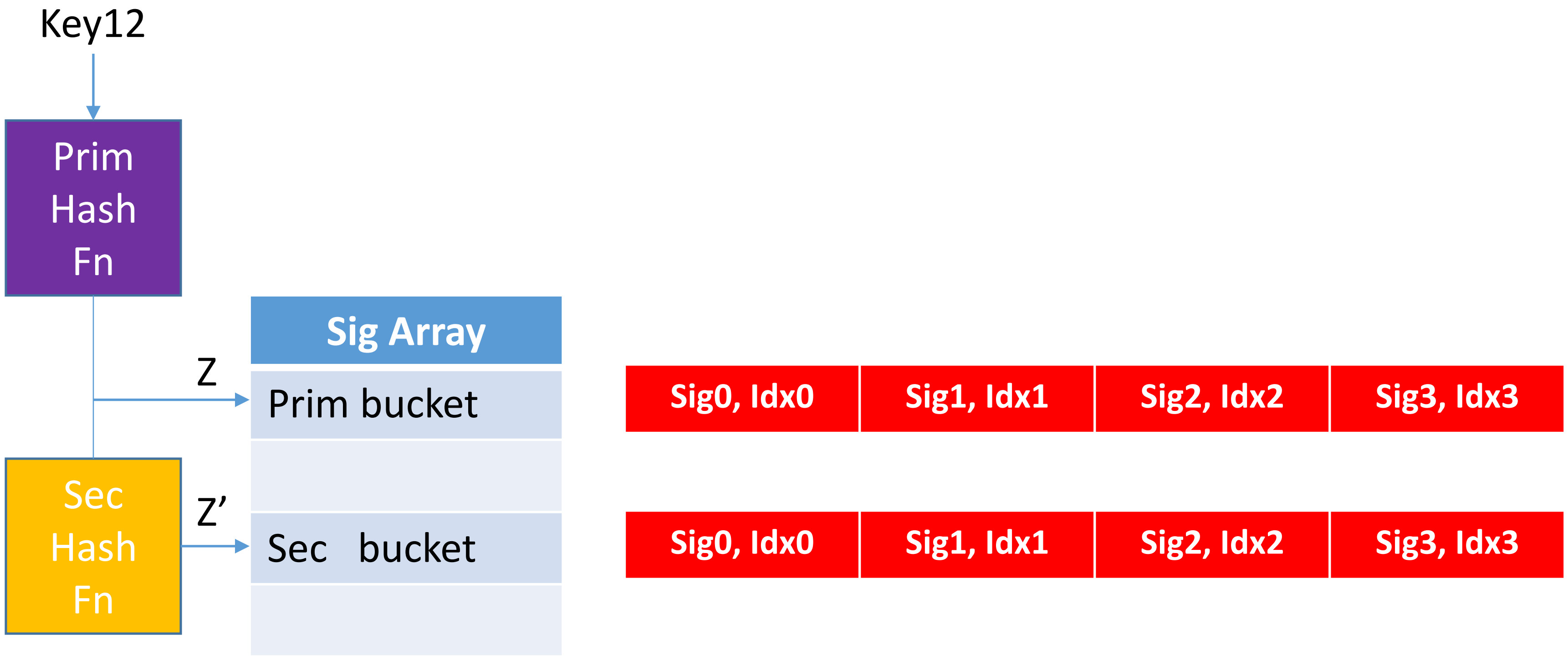
**Sig0, Idx0** | **Sig1, Idx1** | **Sig2, Idx2** | **Sig3, Idx3**

Z’ == Sig3

Return -ENOENT

# DPDK RTE_HASH

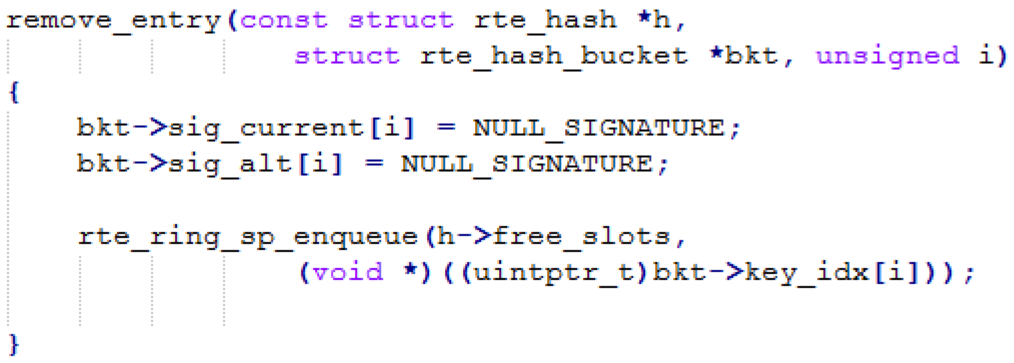
- Lookup



Return -ENOENT

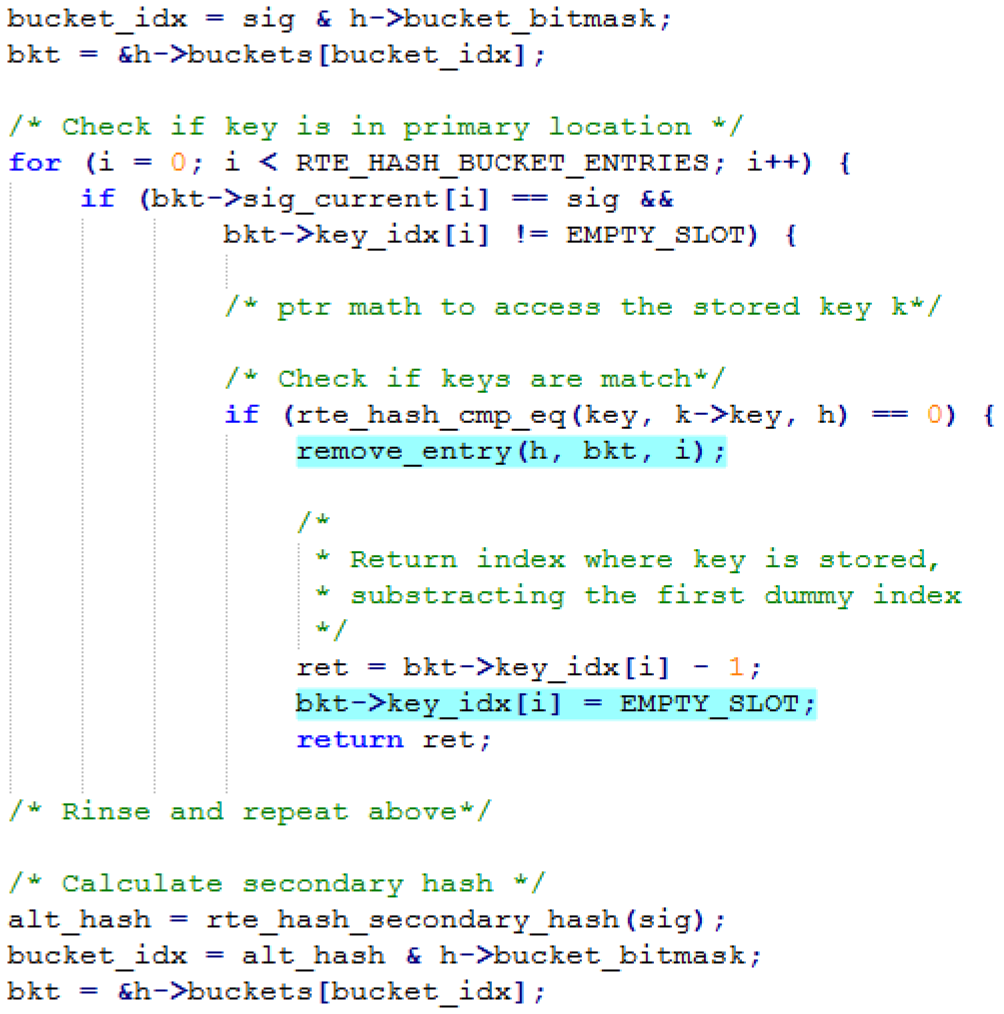
# DPDK RTE_HASH

- Delete



```
remove_entry(const struct rte_hash *h,
            struct rte_hash_bucket *bkt, unsigned i)
{
    bkt->sig_current[i] = NULL_SIGNATURE;
    bkt->sig_alt[i] = NULL_SIGNATURE;

    rte_ring_sp_enqueue(h->free_slots,
            (void *)((uintptr_t)bkt->key_idx[i]));

}
```

Exactly same as lookup. Then remove and enqueue idx.



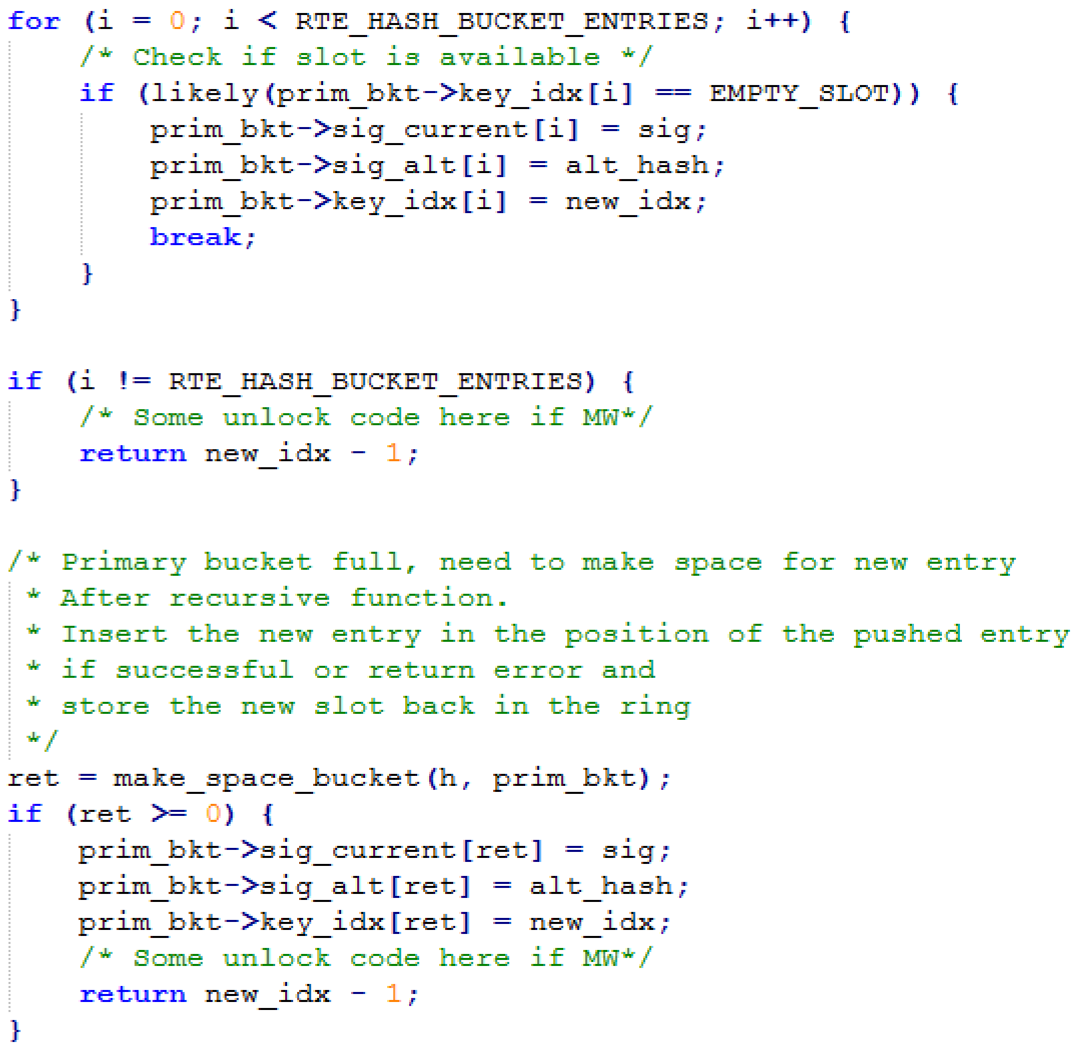
```
bucket_idx = sig & h->bucket_bitmask;
bkt = &h->buckets[bucket_idx];

/* Check if key is in primary location */
for (i = 0; i < RTE_HASH_BUCKET_ENTRIES; i++) {
    if (bkt->sig_current[i] == sig &&
            bkt->key_idx[i] != EMPTY_SLOT) {

            /* ptr math to access the stored key k*/

            /* Check if keys are match*/
            if (rte_hash_cmp_eq(key, k->key, h) == 0) {
                remove_entry(h, bkt, i);

                /*
                 * Return index where key is stored,
                 * substracting the first dummy index
                 */
                ret = bkt->key_idx[i] - 1;
                bkt->key_idx[i] = EMPTY_SLOT;
                return ret;

/* Rinse and repeat above*/

/* Calculate secondary hash */
alt_hash = rte_hash_secondary_hash(sig);
bucket_idx = alt_hash & h->bucket_bitmask;
bkt = &h->buckets[bucket_idx];
```

# DPDK RTE_HASH

- Add
  - Dequeue a slot_id from ring
  - Check if key already exists in primary or secondary location
    - If so update data, enqueue back key_slot and return existing idx
  - Place the key in the key array (slot_id)

# DPDK RTE_HASH

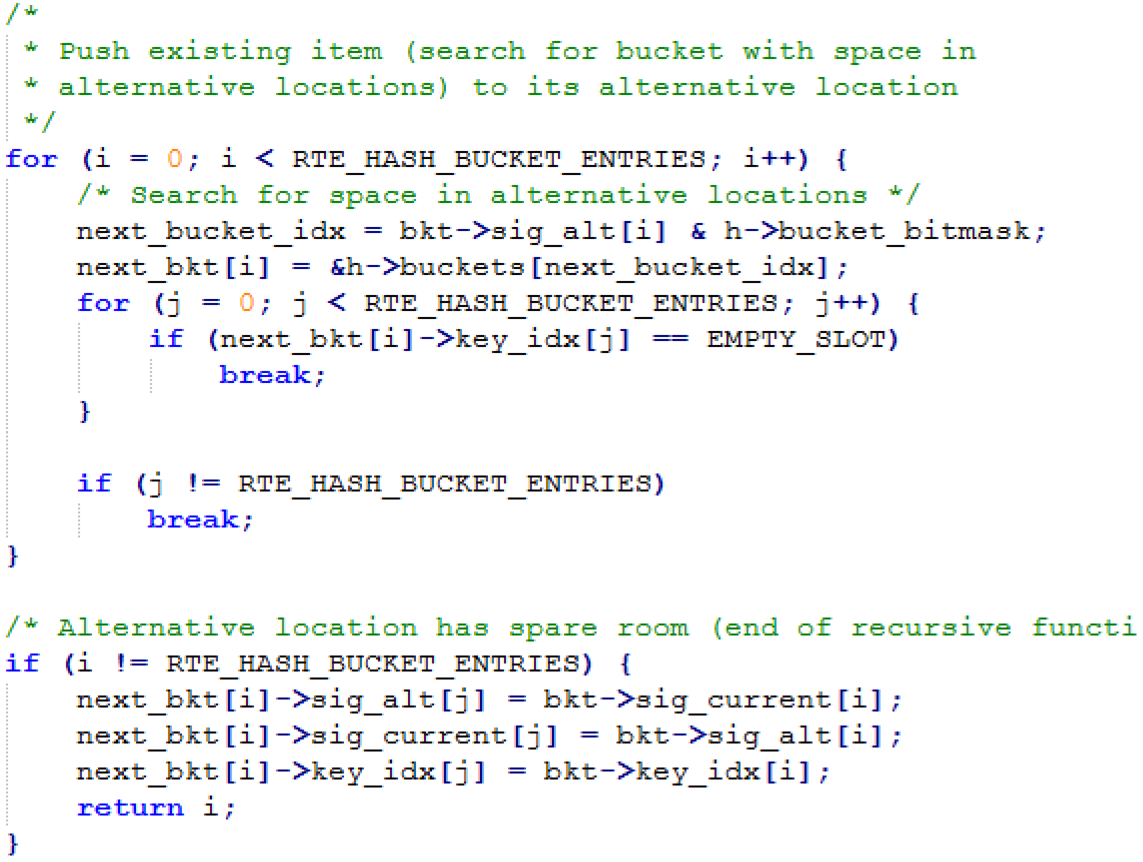
- Add
  - Dequeue a slot_id from ring
  - Check if key already exists in primary or secondary location
    - If so update data, enqueue back key_slot and return existing idx
  - Place the key in the key array (slot_id)
  - See if there is empty entry in Prim bkt
    - If so update sig_cur, sig_alt, idx
    - Return idx
  - Move signatures around to make space for the new key (make_space_bucket)



```
for (i = 0; i < RTE_HASH_BUCKET_ENTRIES; i++) {
    /* Check if slot is available */
    if (likely(prim_bkt->key_idx[i] == EMPTY_SLOT)) {
        prim_bkt->sig_current[i] = sig;
        prim_bkt->sig_alt[i] = alt_hash;
        prim_bkt->key_idx[i] = new_idx;
        break;
    }
}

if (i != RTE_HASH_BUCKET_ENTRIES) {
    /* Some unlock code here if MW*/
    return new_idx - 1;
}

/* Primary bucket full, need to make space for new entry
 * After recursive function.
 * Insert the new entry in the position of the pushed entry
 * if successful or return error and
 * store the new slot back in the ring
 */
ret = make_space_bucket(h, prim_bkt);
if (ret >= 0) {
    prim_bkt->sig_current[ret] = sig;
    prim_bkt->sig_alt[ret] = alt_hash;
    prim_bkt->key_idx[ret] = new_idx;
    /* Some unlock code here if MW*/
    return new_idx - 1;
}
```

# DPDK RTE_HASH

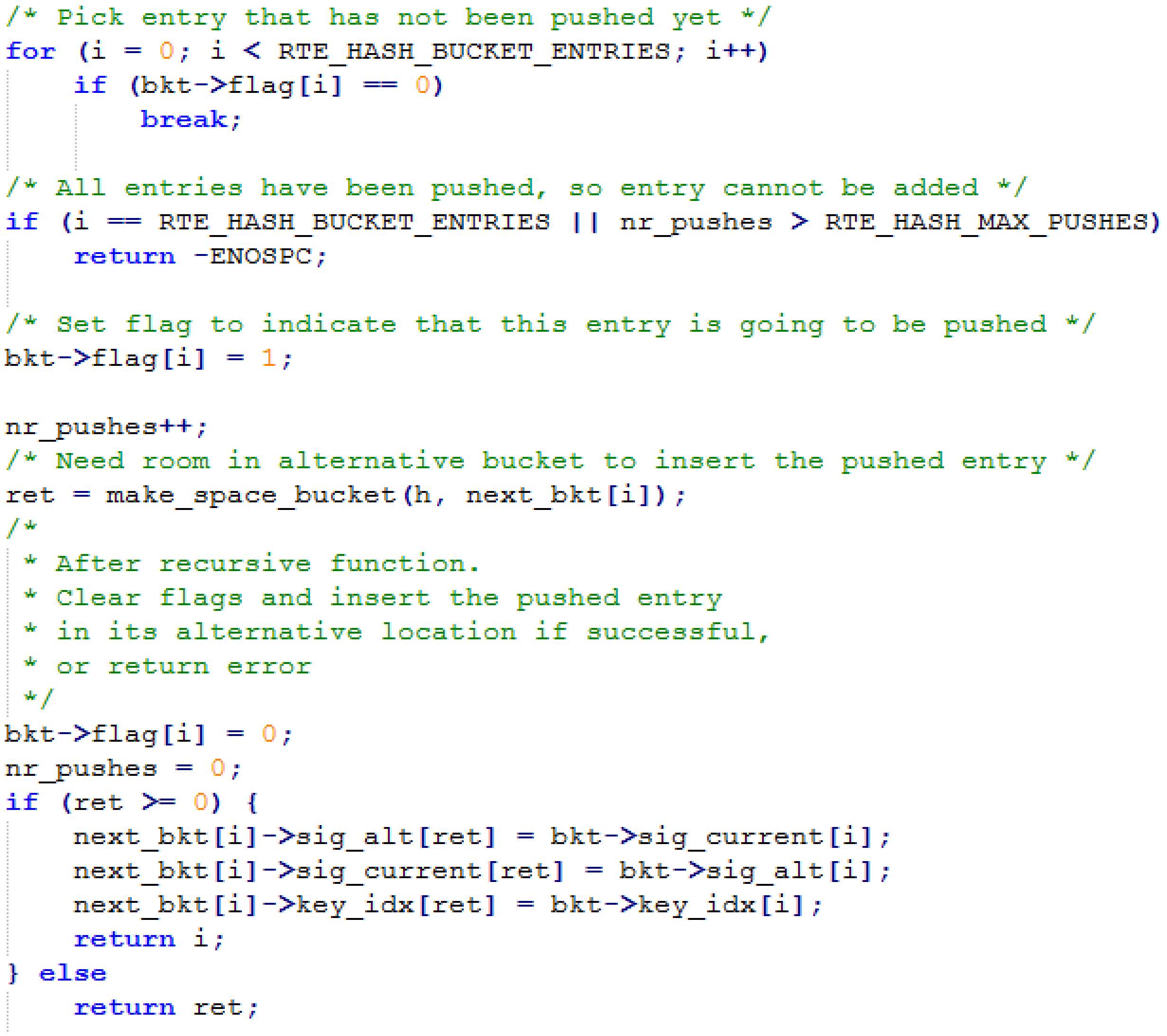
- Add
  - Make_space_bucket(bkt)
    - Iterate over entries in bkt
      - If entry’s secondary location empty, move it
      - We have now made space for an entry. Return the space
    - Recursive cuckoo move



```
/*
 * Push existing item (search for bucket with space in
 * alternative locations) to its alternative location
 */
for (i = 0; i < RTE_HASH_BUCKET_ENTRIES; i++) {
    /* Search for space in alternative locations */
    next_bucket_idx = bkt->sig_alt[i] & h->bucket_bitmask;
    next_bkt[i] = &h->buckets[next_bucket_idx];
    for (j = 0; j < RTE_HASH_BUCKET_ENTRIES; j++) {
        if (next_bkt[i]->key_idx[j] == EMPTY_SLOT)
            break;
    }

    if (j != RTE_HASH_BUCKET_ENTRIES)
        break;
}

/* Alternative location has spare room (end of recursive functi
if (i != RTE_HASH_BUCKET_ENTRIES) {
    next_bkt[i]->sig_alt[j] = bkt->sig_current[i];
    next_bkt[i]->sig_current[j] = bkt->sig_alt[i];
    next_bkt[i]->key_idx[j] = bkt->key_idx[i];
    return i;
}
```

# DPDK RTE_HASH

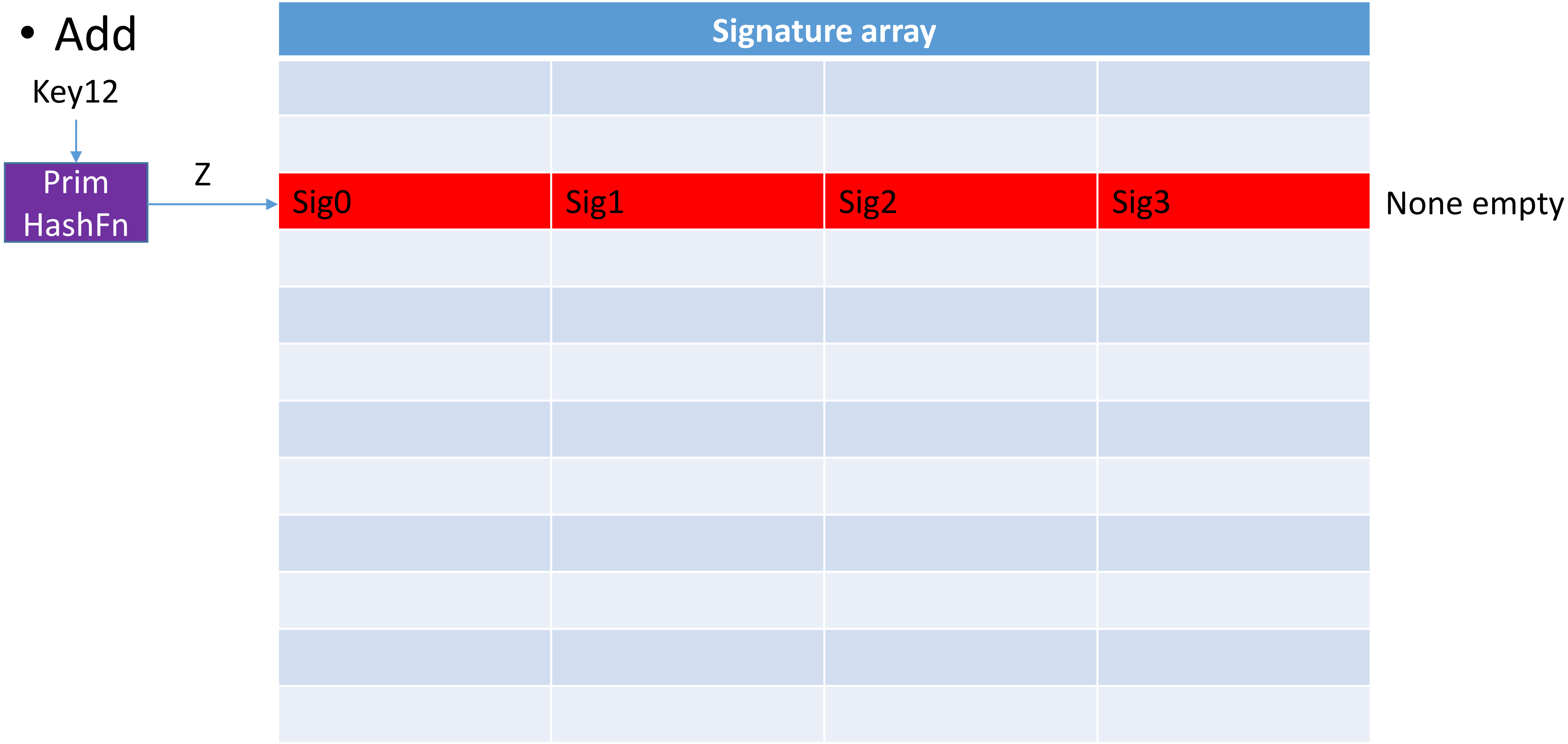
- Add
  - Make_space_bucket(bkt)
    - Iterate over entries in bkt
      - If entry’s secondary location empty, move it
      - We have now made space for an entry. Return the space
    - Pick an victim entry from bkt to move.
    - Space=Make_space_bkt(entry’s alt bkt)
    - Move the entry to Space
    - Return the vacated entry location
      - Caller will use this as its Space



```
/* Pick entry that has not been pushed yet */
for (i = 0; i < RTE_HASH_BUCKET_ENTRIES; i++)
    if (bkt->flag[i] == 0)
        break;

/* All entries have been pushed, so entry cannot be added */
if (i == RTE_HASH_BUCKET_ENTRIES || nr_pushes > RTE_HASH_MAX_PUSHES)
    return -ENOSPC;

/* Set flag to indicate that this entry is going to be pushed */
bkt->flag[i] = 1;

nr_pushes++;
/* Need room in alternative bucket to insert the pushed entry */
ret = make_space_bucket(h, next_bkt[i]);
/*
 * After recursive function.
 * Clear flags and insert the pushed entry
 * in its alternative location if successful,
 * or return error
 */
bkt->flag[i] = 0;
nr_pushes = 0;
if (ret >= 0) {
    next_bkt[i]->sig_alt[ret] = bkt->sig_current[i];
    next_bkt[i]->sig_current[ret] = bkt->sig_alt[i];
    next_bkt[i]->key_idx[ret] = bkt->key_idx[i];
    return i;
} else
    return ret;
```

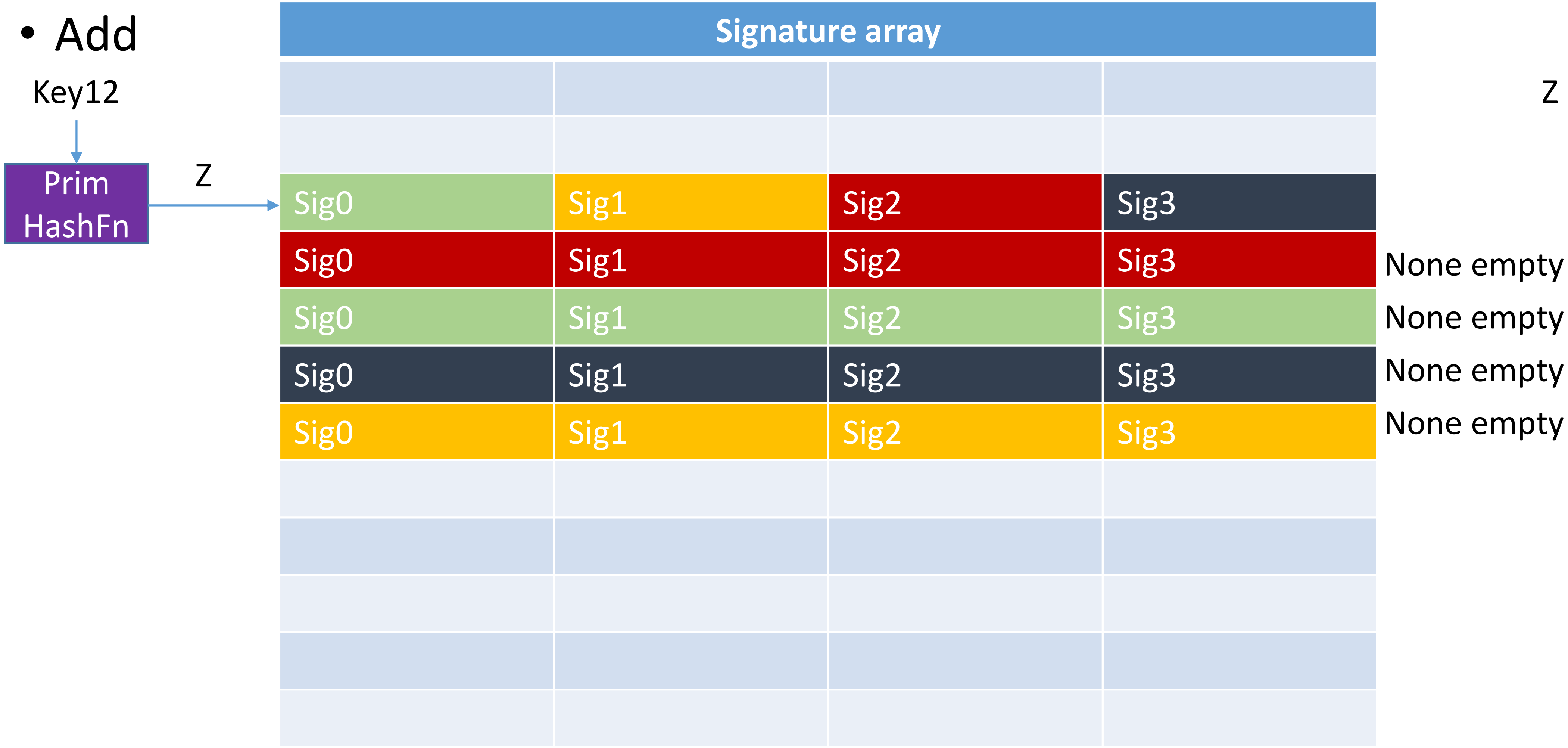
# DPDK RTE_HASH

- Add



# DPDK RTE_HASH

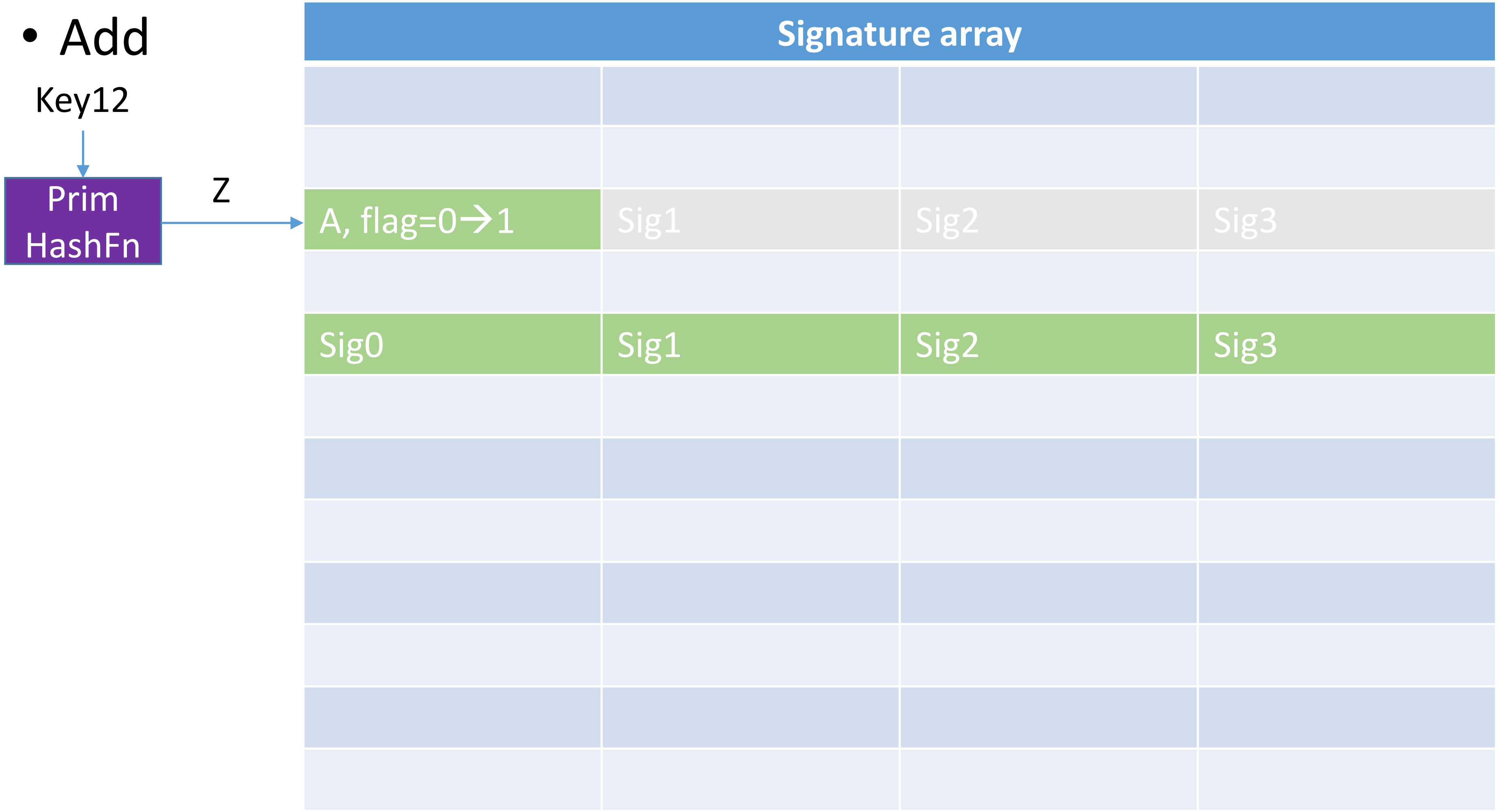
- Add



| Signature array | | | | |
|---|---|---|---|---|
| | | | | Z |
| | | | | |
| Sig0 | Sig1 | Sig2 | Sig3 | |
| Sig0 | Sig1 | Sig2 | Sig3 | None empty |
| Sig0 | Sig1 | Sig2 | Sig3 | None empty |
| Sig0 | Sig1 | Sig2 | Sig3 | None empty |
| Sig0 | Sig1 | Sig2 | Sig3 | None empty |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# DPDK RTE_HASH

- Add

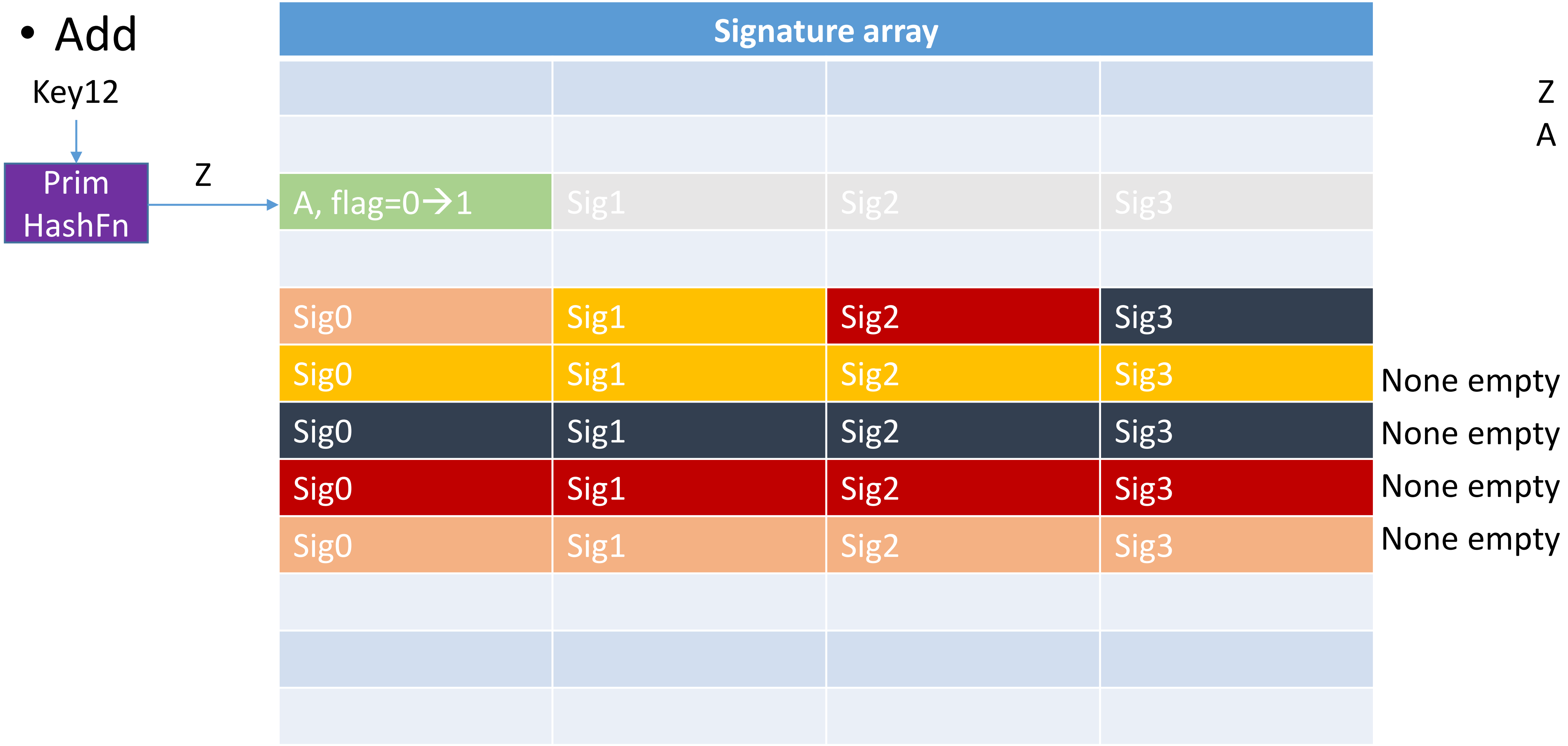


| Signature array | | | |
|---|---|---|---|
| | | | |
| | | | |
| A, flag=0→1 | Sig1 | Sig2 | Sig3 |
| | | | |
| Sig0 | Sig1 | Sig2 | Sig3 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Z
A

# DPDK RTE_HASH

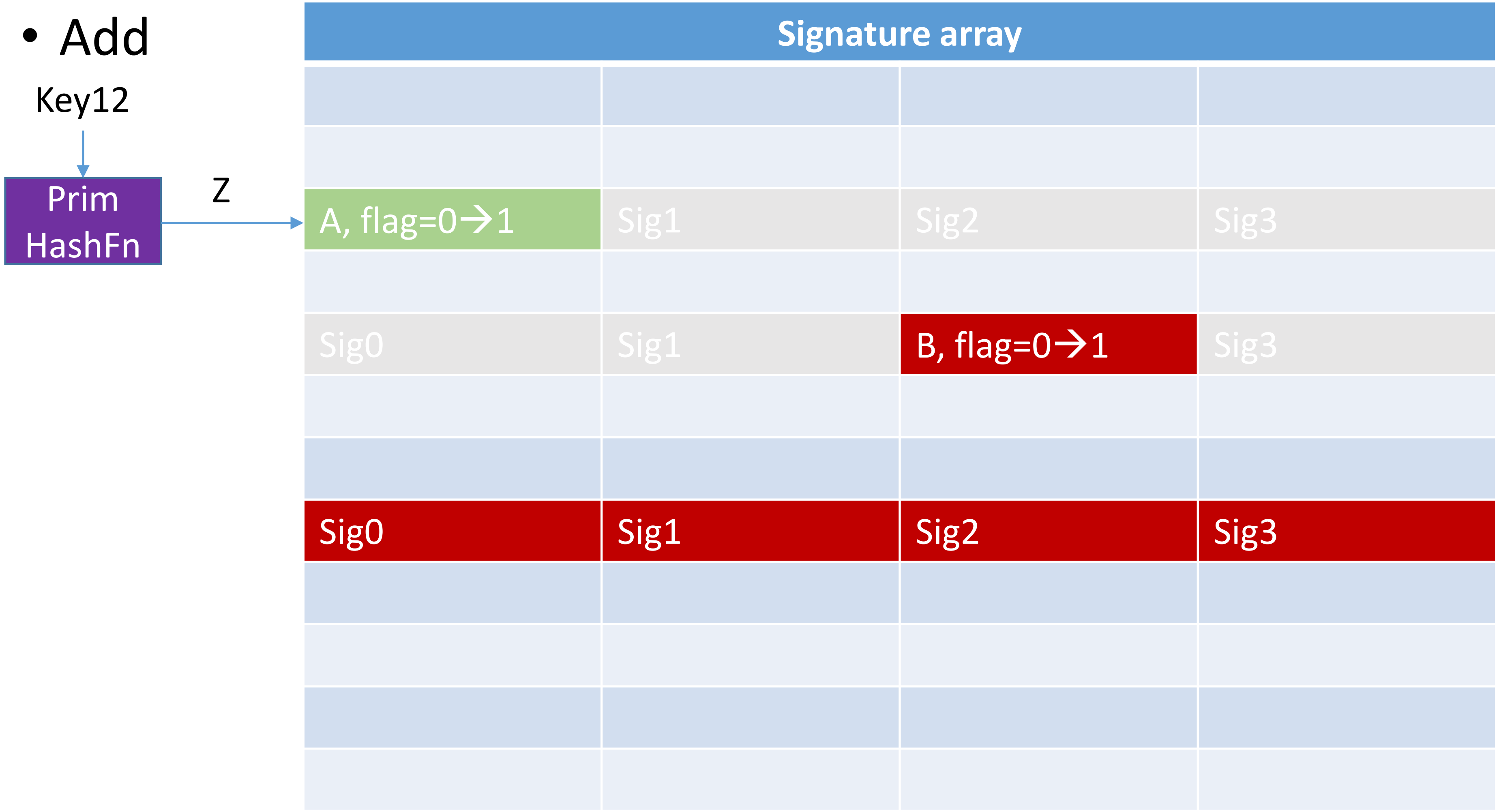
- Add



| Signature array | | | | |
|---|---|---|---|---|
| | | | | Z |
| | | | | A |
| A, flag=0→1 | Sig1 | Sig2 | Sig3 | |
| | | | | |
| Sig0 | Sig1 | Sig2 | Sig3 | |
| Sig0 | Sig1 | Sig2 | Sig3 | None empty |
| Sig0 | Sig1 | Sig2 | Sig3 | None empty |
| Sig0 | Sig1 | Sig2 | Sig3 | None empty |
| Sig0 | Sig1 | Sig2 | Sig3 | None empty |
| | | | | |
| | | | | |
| | | | | |

# DPDK RTE_HASH

- Add

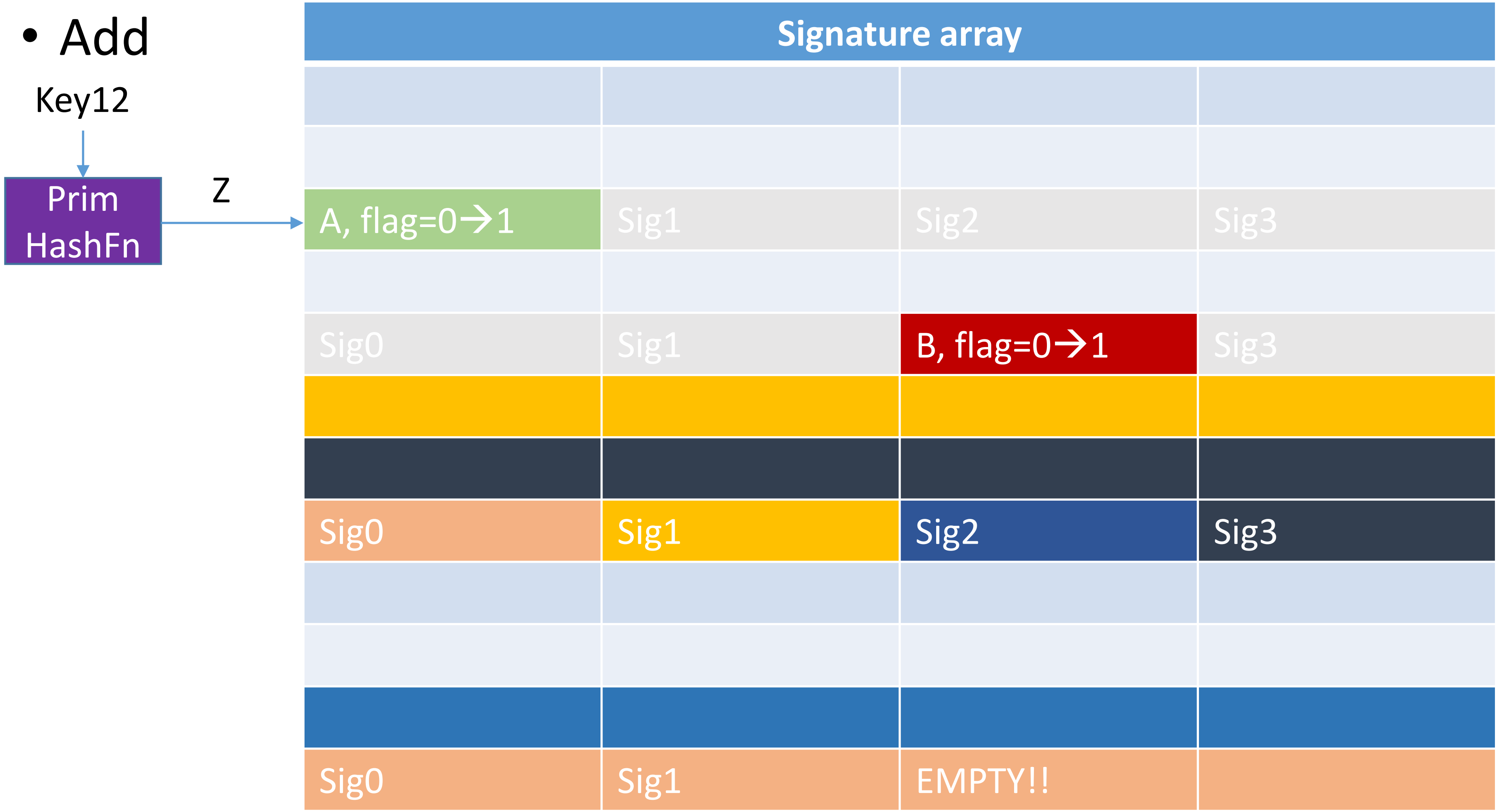


| Signature array | | | |
|---|---|---|---|
| | | | |
| | | | |
| A, flag=0→1 | Sig1 | Sig2 | Sig3 |
| | | | |
| Sig0 | Sig1 | B, flag=0→1 | Sig3 |
| | | | |
| | | | |
| Sig0 | Sig1 | Sig2 | Sig3 |
| | | | |
| | | | |
| | | | |
| | | | |

Z
A
B

# DPDK RTE_HASH

- Add

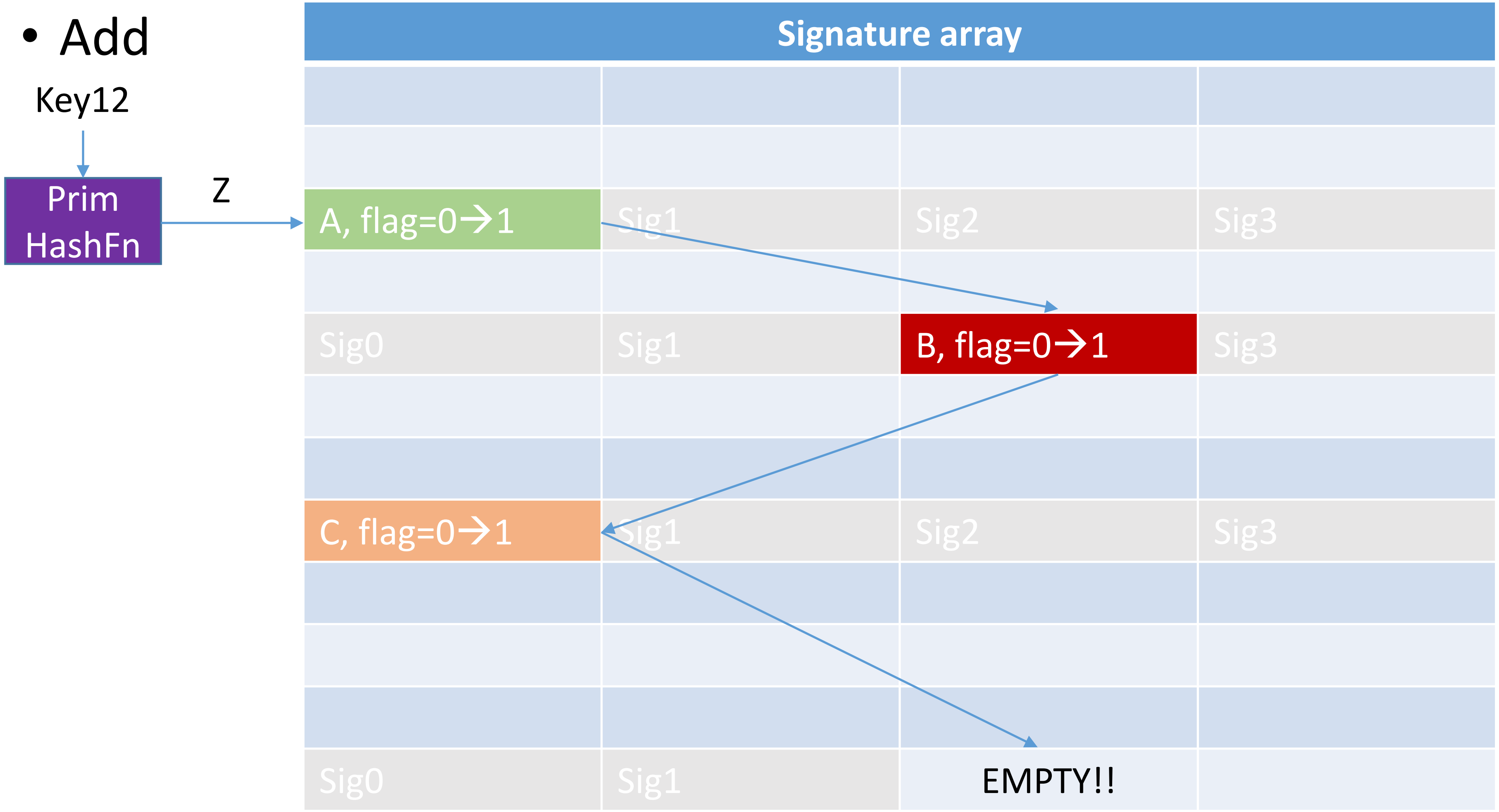


| Signature array | | | |
|---|---|---|---|
| | | | |
| | | | |
| A, flag=0→1 | Sig1 | Sig2 | Sig3 |
| | | | |
| Sig0 | Sig1 | B, flag=0→1 | Sig3 |
| | | | |
| | | | |
| Sig0 | Sig1 | Sig2 | Sig3 |
| | | | |
| | | | |
| | | | |
| Sig0 | Sig1 | EMPTY!! | |

Z
A
B

# DPDK RTE_HASH

- Add



Key12

Prim HashFn

Z

Signature array

A, flag=0→1

Sig1

Sig2

Sig3

Sig0

Sig1

B, flag=0→1

Sig3

C, flag=0→1

Sig1

Sig2

Sig3

Sig0

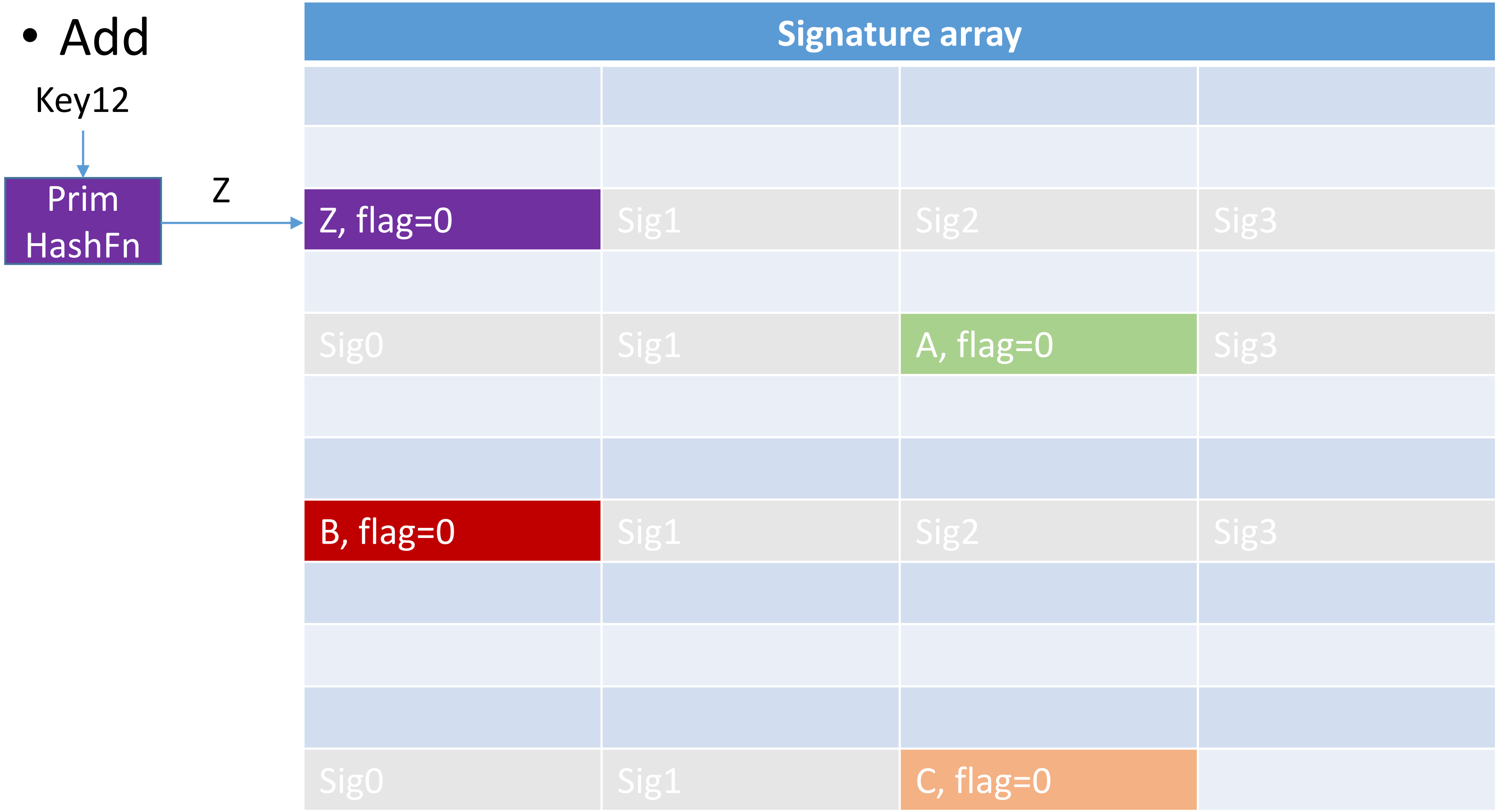
Sig1

EMPTY!!

Z
A
B
C
EMPTY

# DPDK RTE_HASH

- Add



| Signature array | | | |
|---|---|---|---|
| | | | |
| | | | |
| Z, flag=0 | Sig1 | Sig2 | Sig3 |
| | | | |
| Sig0 | Sig1 | A, flag=0 | Sig3 |
| | | | |
| | | | |
| B, flag=0 | Sig1 | Sig2 | Sig3 |
| | | | |
| | | | |
| | | | |
| Sig0 | Sig1 | C, flag=0 | |

# Sections

- Hash and Hash-Table
  - 101
- Skeleton App
  - Extend this app to filter packets by using the Hash library
- DPDK RTE_HASH
  - Library internals
- Test App
  - Unit test and perf tests. Look at hash related tests
- Benchmark changes to lib
  - Modify Add procedure. Use test app to check the effect

# Test App  - Microbench

- hash_functions_autotest
    - Measure cycles for hashing
    - jhash vs rte_hash_crc
    - For different Key lengths, seeds

- hash_autotest
    - Unit tests
    - Average Table Utilization %

- hash_perf_autotest
    - Measure cycles for Add, Lookup, Lookup_bulk, Delete
    - w/(o) pre-computed hash values
    - For different Key lengths

# Test App - Microbench

- hash_scaling_autotest
  - Measure cycles for multi-writer Add
  - w/(o) Hardware transactional memory
  - No lookup or sanity check

- hash_multiwriter_autotest
  - Similar w/ checks for duplicated and lost keys

# Test App  - Microbench

```
sudo ./build/app/test --no-pci -- -i
RTE>>hash_functions_autotest
RTE>>hash_autotest
RTE>>hash_perf_autotest
RTE>>hash_scaling_autotest
RTE>>hash_multiwriter_autotest
```

# Sections

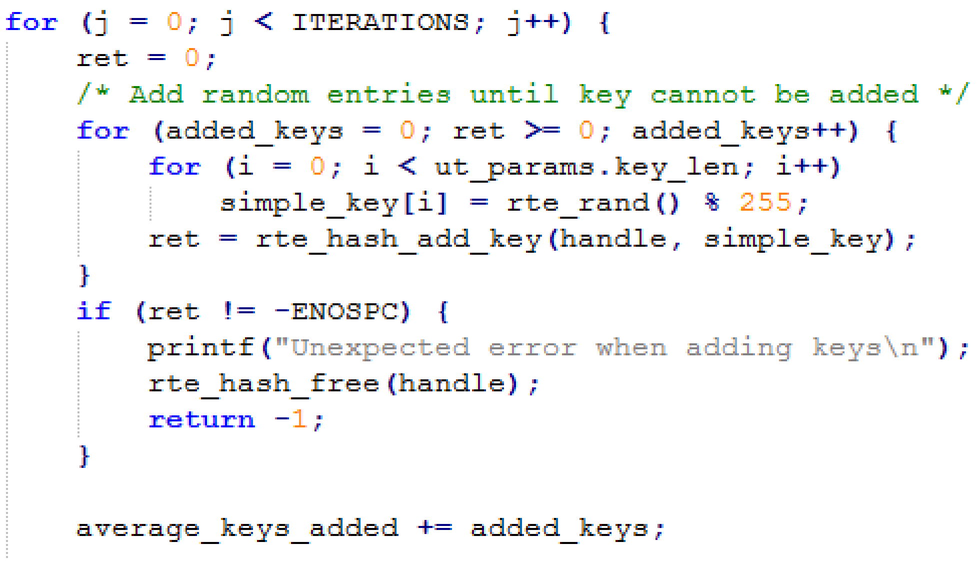
- Hash and Hash-Table
  - 101
- Skeleton App
  - Extend this app to filter packets by using the Hash library
- DPDK RTE_HASH
  - Library internals
- Test App
  - Unit test and perf tests. Look at hash related tests
- Benchmark changes to lib
  - Modify Add procedure. Use test app to check the effect

# Benchmark changes to lib

ADD related changes. Affects utilization

So hash_autotest

- Primary only
- Secondary hash = Primary + 1
  - w/(o) recursive cuckoo move
- Secondary hash
  - w/(o) recursive cuckoo move



```
for (j = 0; j < ITERATIONS; j++) {
    ret = 0;
    /* Add random entries until key cannot be added */
    for (added_keys = 0; ret >= 0; added_keys++) {
        for (i = 0; i < ut_params.key_len; i++)
            simple_key[i] = rte_rand() % 255;
        ret = rte_hash_add_key(handle, simple_key);
    }
    if (ret != -ENOSPC) {
        printf("Unexpected error when adding keys\n");
        rte_hash_free(handle);
        return -1;
    }

    average_keys_added += added_keys;
```

# Benchmark changes to lib

ADD related changes. Affects utilization

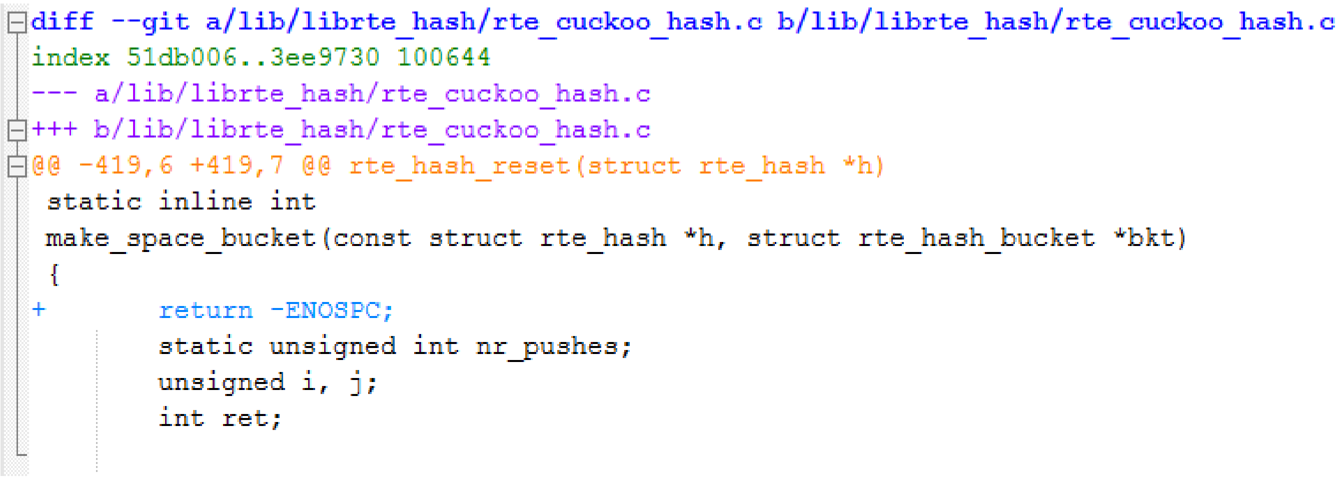
So hash_autotest

```
git checkout lib/librte_hash/rte_cuckoo_hash.c
git apply /vagrant/patches/test_<>.patch
make -j > /dev/null
sudo ./build/app/test --no-pci -- -i
hash_autotest
```

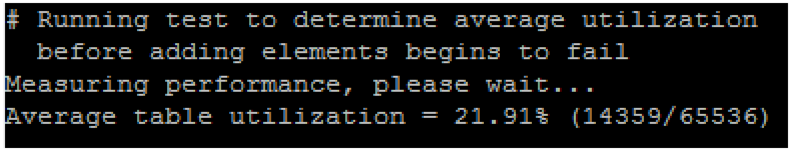
# Benchmark changes to lib

## hash_autotest (test_primary_only.patch)

- Primary only –
  - If there is no space in primary bucket, make_space_bucket is called.
  - So return -ENOSPC immediately to emulate a primary only scenario



```
diff --git a/lib/librte_hash/rte_cuckoo_hash.c b/lib/librte_hash/rte_cuckoo_hash.c
index 51db006..3ee9730 100644
--- a/lib/librte_hash/rte_cuckoo_hash.c
+++ b/lib/librte_hash/rte_cuckoo_hash.c
@@ -419,6 +419,7 @@ rte_hash_reset(struct rte_hash *h)
 static inline int
 make_space_bucket(const struct rte_hash *h, struct rte_hash_bucket *bkt)
 {
+        return -ENOSPC;
         static unsigned int nr_pushes;
         unsigned i, j;
         int ret;
```

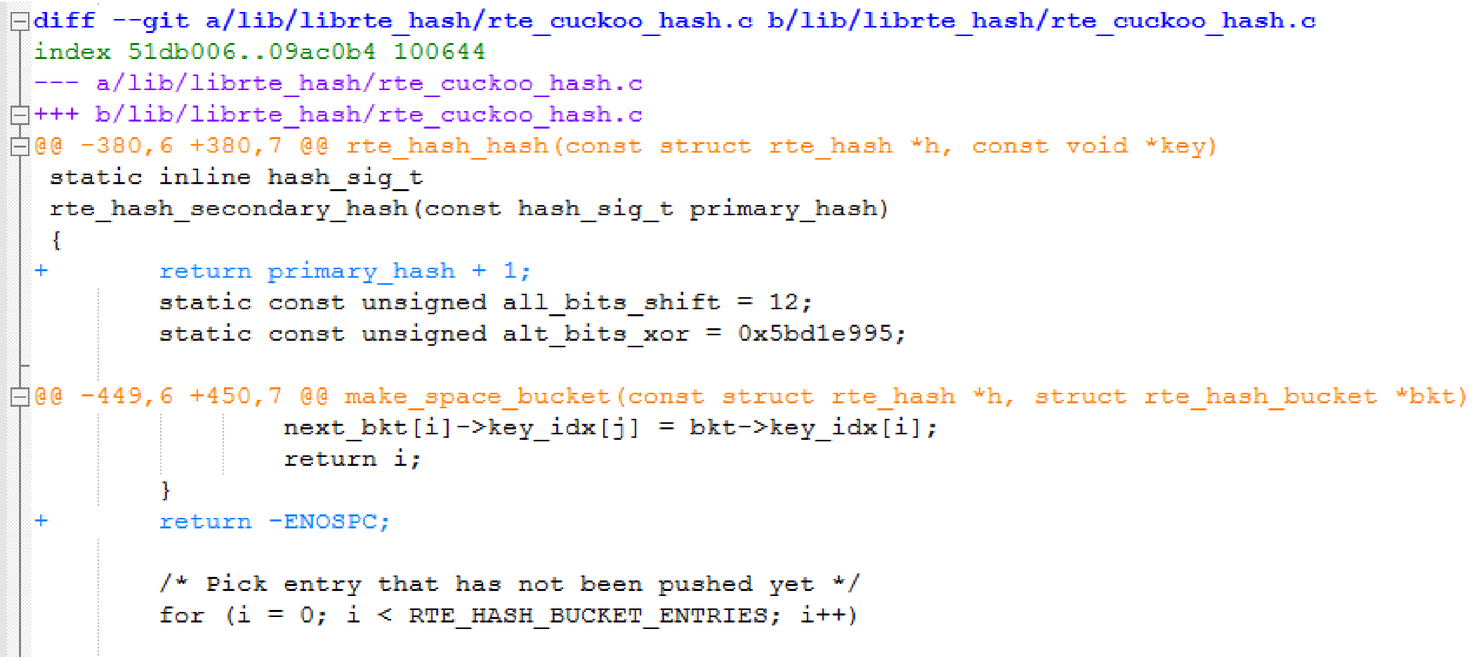


```
# Running test to determine average utilization
  before adding elements begins to fail
Measuring performance, please wait...
Average table utilization = 21.91% (14359/65536)
```

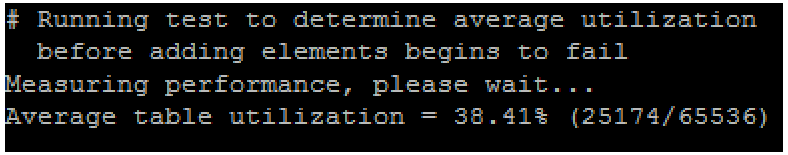
# Benchmark changes to lib

## hash_autotest (test_prim_plus_one_wo_rcuckoo.patch)

- Secondary hash = Primary hash + 1
  - w/o recursive cuckoo move



```
diff --git a/lib/librte_hash/rte_cuckoo_hash.c b/lib/librte_hash/rte_cuckoo_hash.c
index 51db006..09ac0b4 100644
--- a/lib/librte_hash/rte_cuckoo_hash.c
+++ b/lib/librte_hash/rte_cuckoo_hash.c
@@ -380,6 +380,7 @@ rte_hash_hash(const struct rte_hash *h, const void *key)
 static inline hash_sig_t
 rte_hash_secondary_hash(const hash_sig_t primary_hash)
 {
+        return primary_hash + 1;
         static const unsigned all_bits_shift = 12;
         static const unsigned alt_bits_xor = 0x5bd1e995;

@@ -449,6 +450,7 @@ make_space_bucket(const struct rte_hash *h, struct rte_hash_bucket *bkt)
                         next_bkt[i]->key_idx[j] = bkt->key_idx[i];
                         return i;
                 }
+        return -ENOSPC;

         /* Pick entry that has not been pushed yet */
         for (i = 0; i < RTE_HASH_BUCKET_ENTRIES; i++)
```

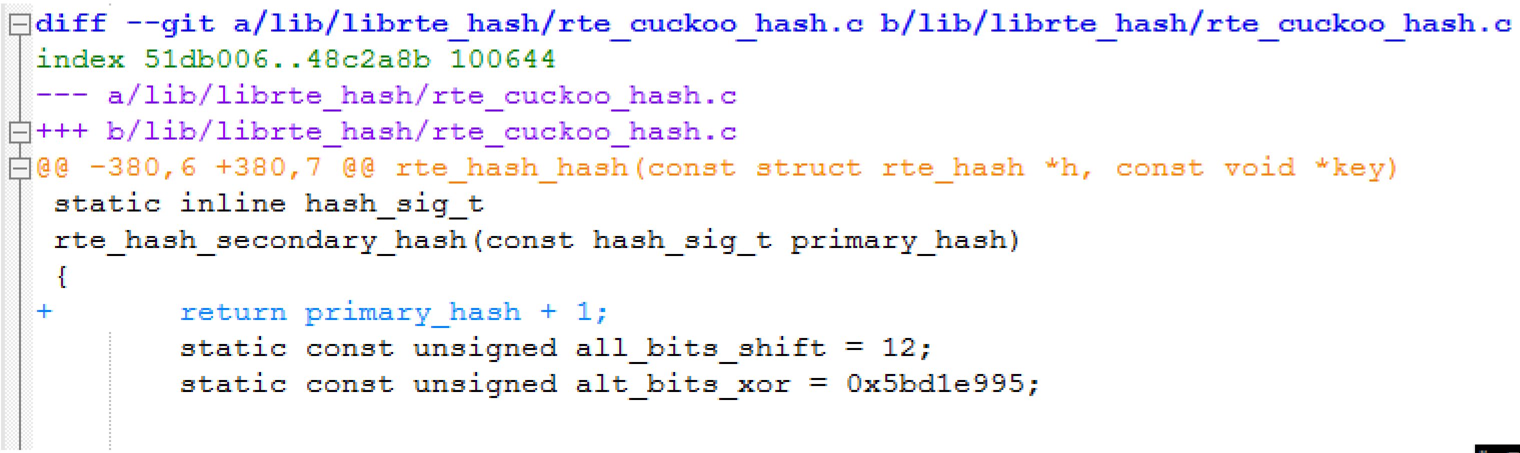


```
# Running test to determine average utilization
  before adding elements begins to fail
Measuring performance, please wait...
Average table utilization = 38.41% (25174/65536)
```

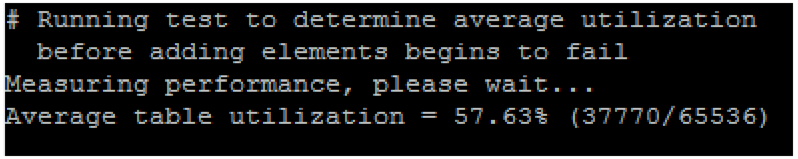
# Benchmark changes to lib

## hash_autotest (test_prim_plus_one_w_rcuckoo.patch)

- Secondary hash = Primary hash + 1
  - w/ recursive cuckoo move



```
diff --git a/lib/librte_hash/rte_cuckoo_hash.c b/lib/librte_hash/rte_cuckoo_hash.c
index 51db006..48c2a8b 100644
--- a/lib/librte_hash/rte_cuckoo_hash.c
+++ b/lib/librte_hash/rte_cuckoo_hash.c
@@ -380,6 +380,7 @@ rte_hash_hash(const struct rte_hash *h, const void *key)
 static inline hash_sig_t
 rte_hash_secondary_hash(const hash_sig_t primary_hash)
 {
+	return primary_hash + 1;
 	static const unsigned all_bits_shift = 12;
 	static const unsigned alt_bits_xor = 0x5bd1e995;
```

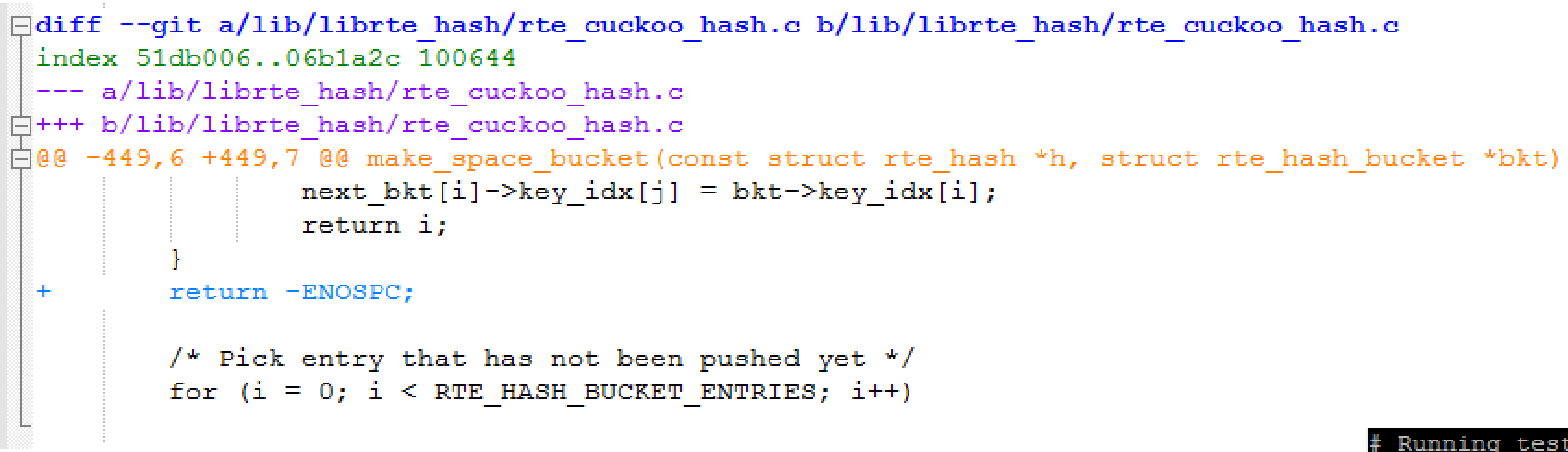


```
# Running test to determine average utilization
  before adding elements begins to fail
Measuring performance, please wait...
Average table utilization = 57.63% (37770/65536)
```

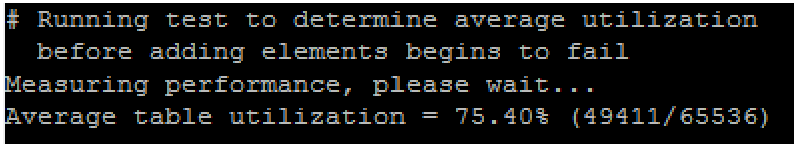
# Benchmark changes to lib

## hash_autotest (test_sec_wo_rcuckoo.patch)

- Secondary hash as-is
  - w/o recursive cuckoo move



```
diff --git a/lib/librte_hash/rte_cuckoo_hash.c b/lib/librte_hash/rte_cuckoo_hash.c
index 51db006..06b1a2c 100644
--- a/lib/librte_hash/rte_cuckoo_hash.c
+++ b/lib/librte_hash/rte_cuckoo_hash.c
@@ -449,6 +449,7 @@ make_space_bucket(const struct rte_hash *h, struct rte_hash_bucket *bkt)
                next_bkt[i]->key_idx[j] = bkt->key_idx[i];
                return i;
        }
+       return -ENOSPC;

        /* Pick entry that has not been pushed yet */
        for (i = 0; i < RTE_HASH_BUCKET_ENTRIES; i++)
```

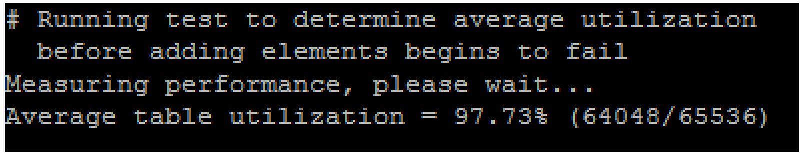


```
# Running test to determine average utilization
  before adding elements begins to fail
Measuring performance, please wait...
Average table utilization = 75.40% (49411/65536)
```

# Benchmark changes to lib

hash_autotest

- Secondary hash as-is
    - w/ recursive cuckoo move



```
# Running test to determine average utilization
  before adding elements begins to fail
Measuring performance, please wait...
Average table utilization = 97.73% (64048/65536)
```

# Benchmark changes to lib

hash_perf_autotest – Assignment

- Compare Lookup performance results of different changes
- Decide trade-offs between util and lookup speed

# BACKUP

# PCAP generation

```
pip install --user scapy
```

```
from scapy.all import *
import struct
import socket
pkts = []

for i in range(1<<16):
        pkts.append(Ether(dst='de:ad:be:ef:ab:cd')/IP(src='1.2.3.4',
dst=socket.inet_ntoa(struct.pack("!I", i)))/UDP(sport=1234,dport=5678)/"OutOfTheBoxNetDevs")

wrpcap('64K_dst0.pcap', pkts)
wrpcap('64K_dst1.pcap', pkts)
```

# DPDK install

```
git clone http://dpdk.org/git/dpdk
cd dpdk
git checkout v16.11

#Enable PCAP PMD
sed -ri 's,(PMD_PCAP=).*,\1y,' config/common_base

make config T=x86_64-native-linuxapp-gcc
make -j4

export RTE_SDK=$(pwd)
export RTE_TARGET=build
```