EBtree

Design for a scheduler, and use (almost) everywhere

Andjelko Iharos
aiharos@haproxy.com

QCon New York, June 24-26, 2019

EBtree

Design for a scheduler, and use (almost) everywhere

Andjelko Iharos
aiharos@haproxy.com

QCon New York, June 24-26, 2019

EBtree features

- Fast tree descent & search
- Memory efficient
- Lookup by mask or prefix (i.e. IPv4 and IPv6)
- Optimized for inserts and deletes
- Great with bit-addressable data

Outline

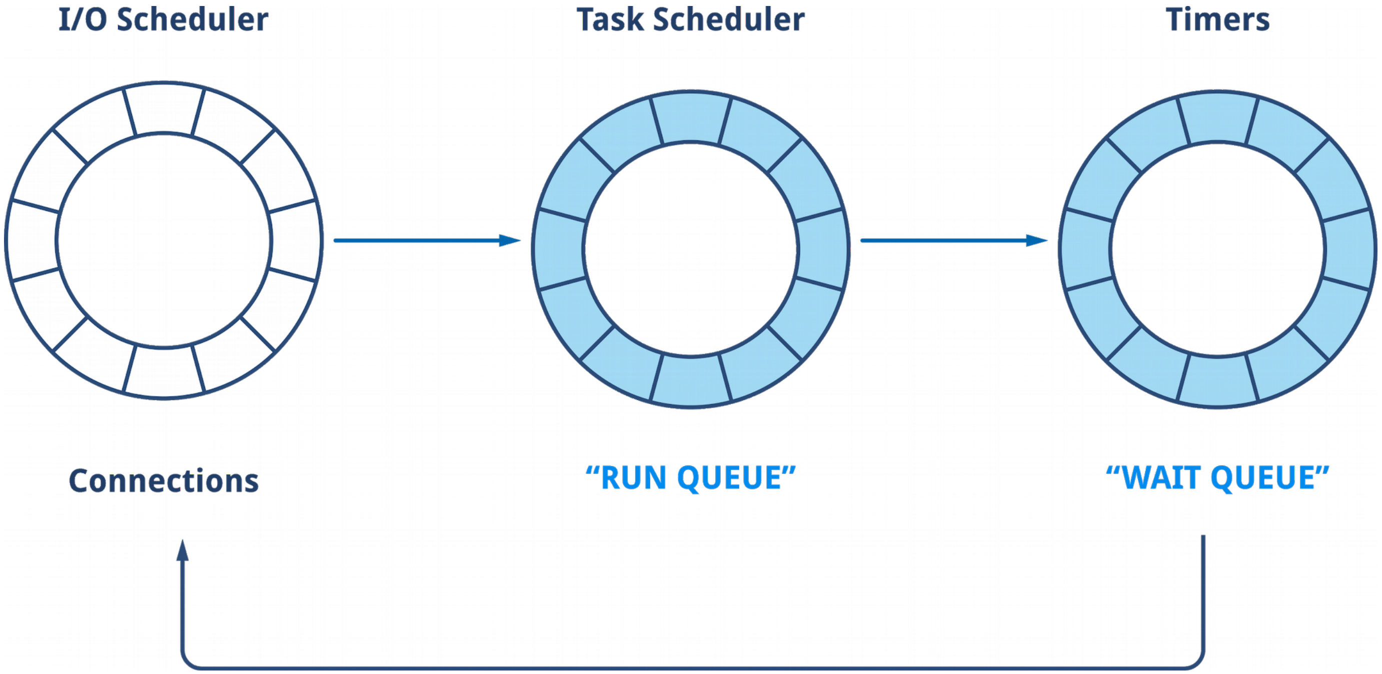
- Scheduling requirements
- Candidate solutions
- EBtree design
- Implementation
- Production use
- Results

Scheduling requirements

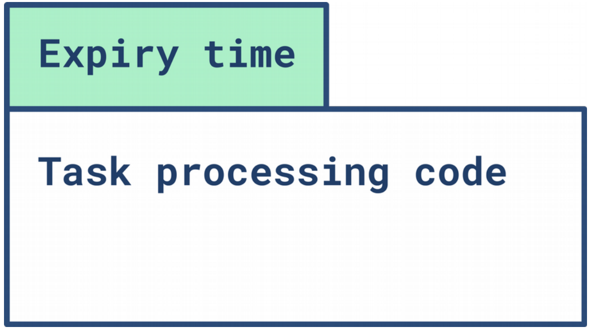
HAProxy event loop

- Handle network connections
- Run active tasks
- Check suspended tasks, wake them up

HAProxy event loop



HAProxy task

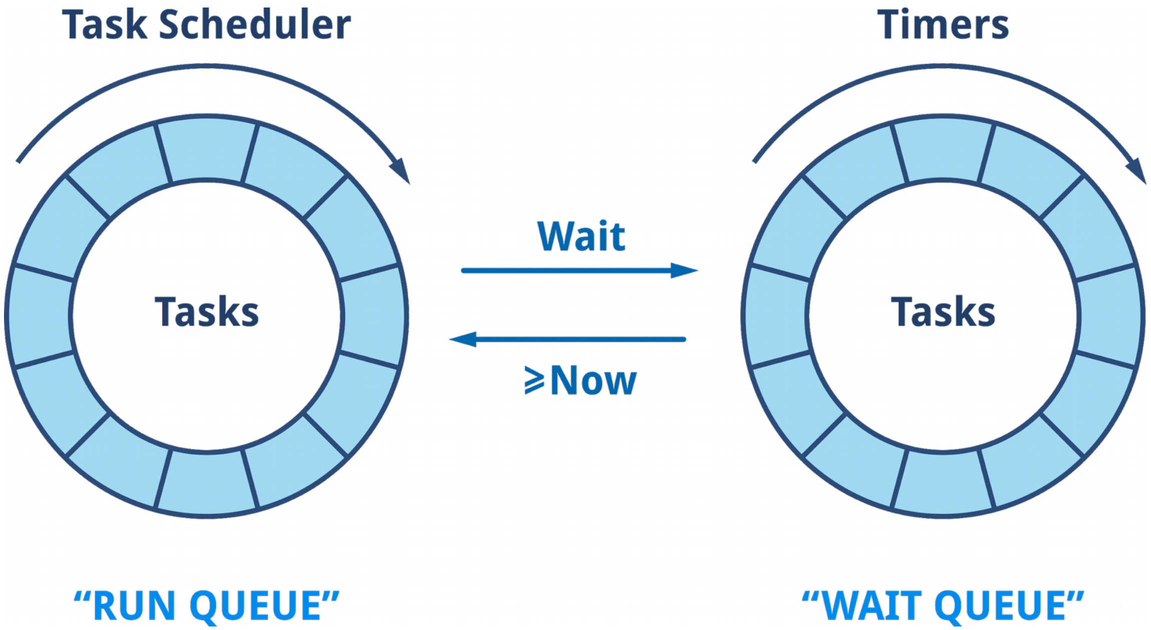


Expiry time

Task processing code

Scheduler features

- Active & suspended tasks
- Insert
- Duplicates
- Read sorted
- Delete
- Priorities



Task Scheduler

Timers

Tasks

Wait

≥Now

Tasks

“RUN QUEUE”

“WAIT QUEUE”

Scheduling environment

- Up to high frequency of events
- Up to very large number of entries
- Large variations in rate of entry change
- Frequent lookups

Desirable qualities

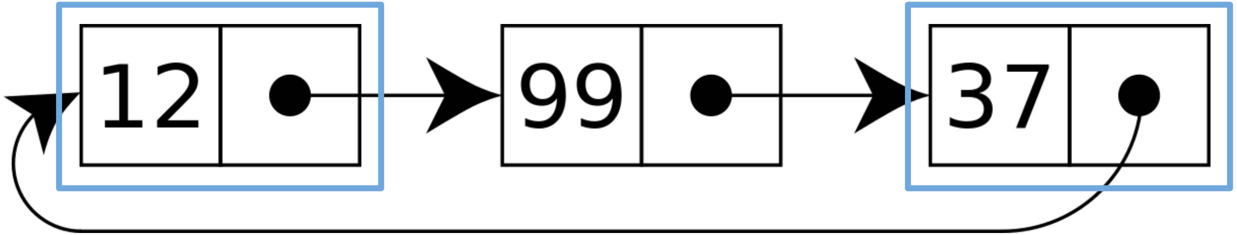
- Speed
- Predictability
- Simplicity

Candidate solutions

Basic data structures

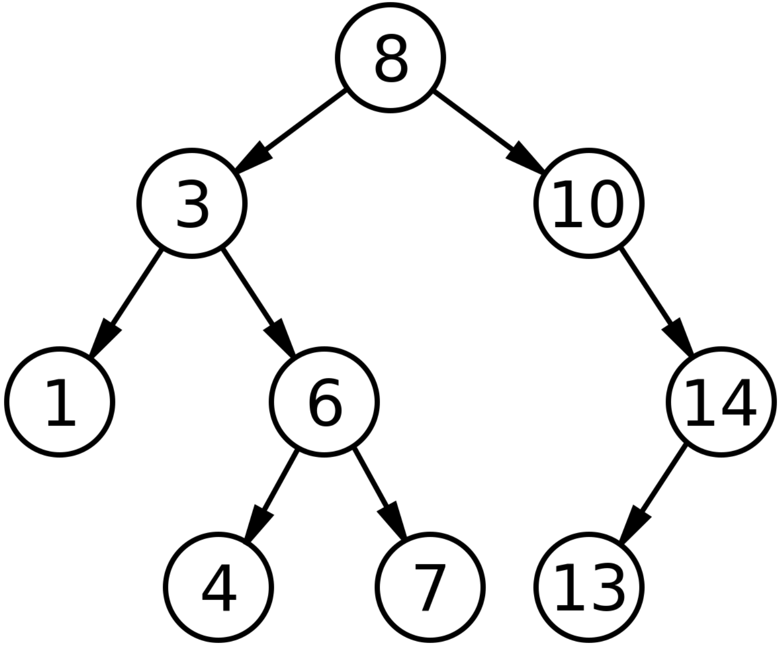
- Array
- Linked list
- Stack, Queue
- Hash Map
- Tree

Linked list



By Lasindi - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=2194027

Binary search tree

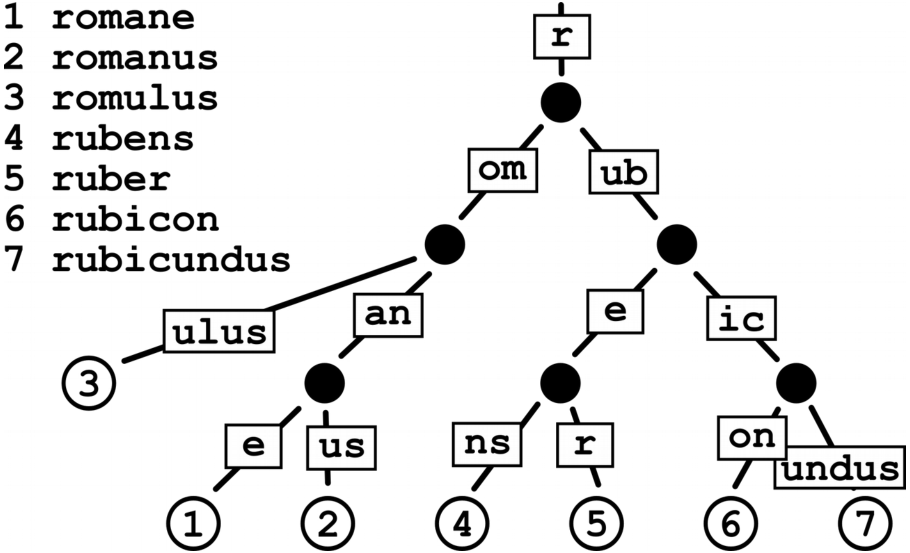


By No machine-readable author provided. Dcoetzee assumed (based on copyright claims). - No machine-readable source provided. Own work assumed (based on copyright claims)., Public Domain, https://commons.wikimedia.org/w/index.php?curid=488330

AVL tree rotations

By Bruno Schalch - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=64250599

Prefix (Radix) trees



1 romane
2 romanus
3 romulus
4 rubens
5 ruber
6 rubicon
7 rubicundus

By Claudio Rocchini - Own work, CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=2118795

Prefix (Radix) trees

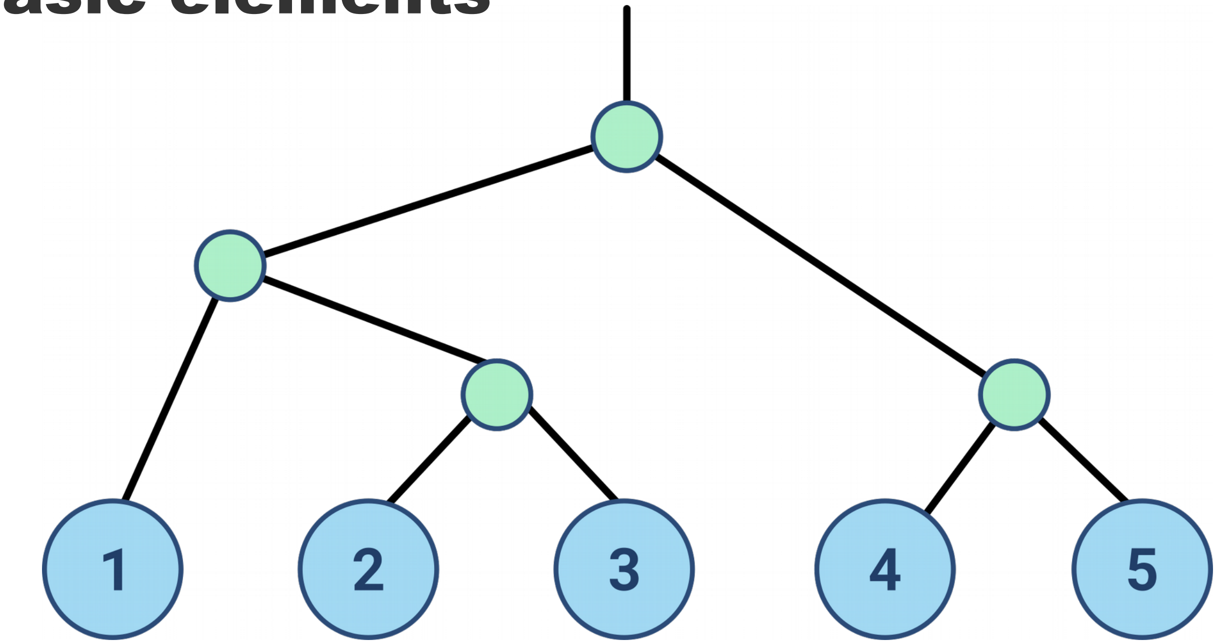
- O(log n) insert, O(1) delete
- Fast comparison even for long keys
- Prefix matching
- Nodes and leaves are different
- Not balanced

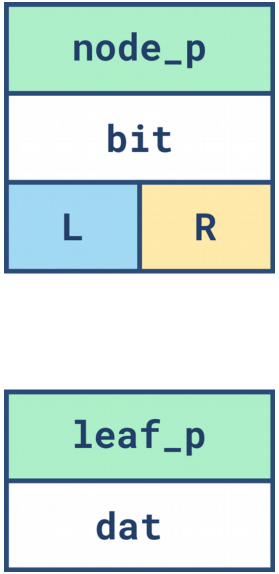
EBtree design

Can we improve?

- Simplify memory management
- Reduce impact of imbalance and tree height

Basic elements





node_p

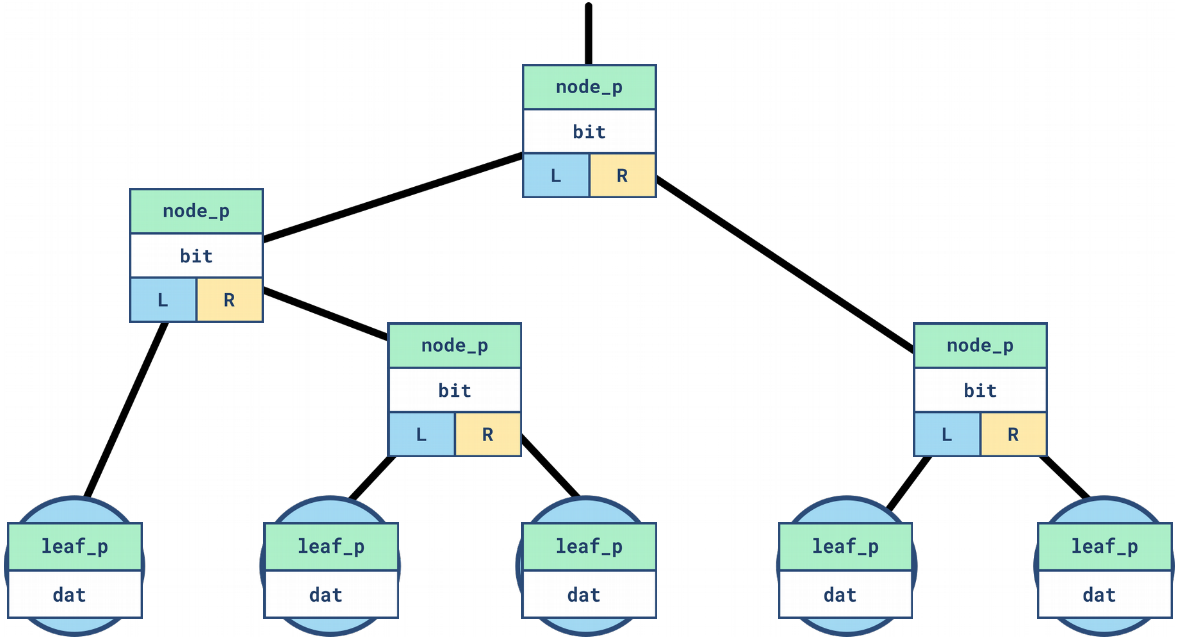
bit

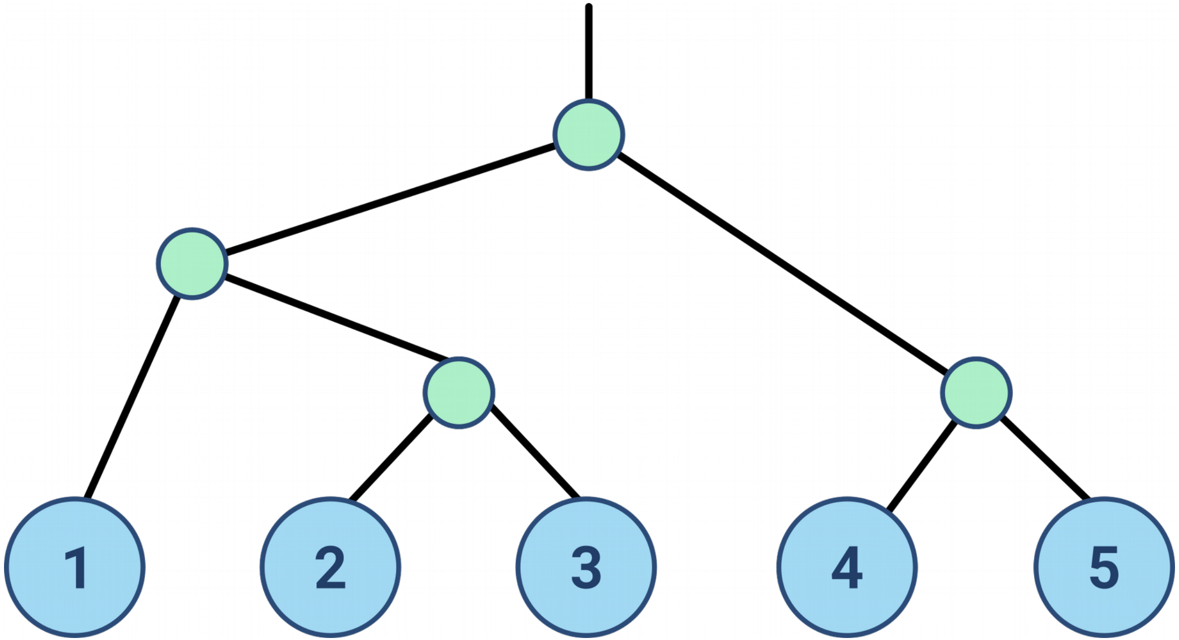
L

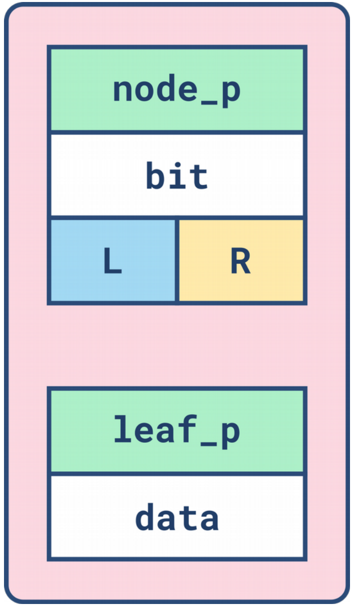
R

leaf_p

dat


node_p
bit
L
R
node_p
bit
L
R
node_p
bit
L
R
node_p
bit
L
R
leaf_p
dat
leaf_p
dat
leaf_p
dat
leaf_p
dat
leaf_p
dat


1
2
3
4
5



node_p

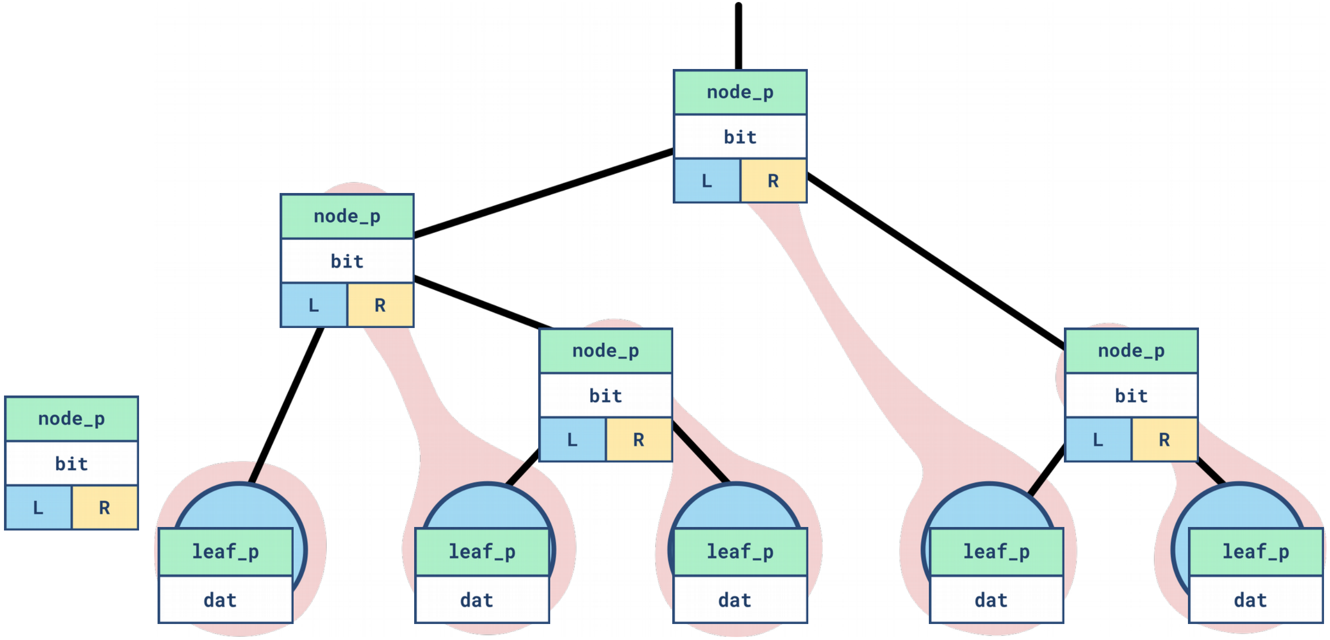
bit

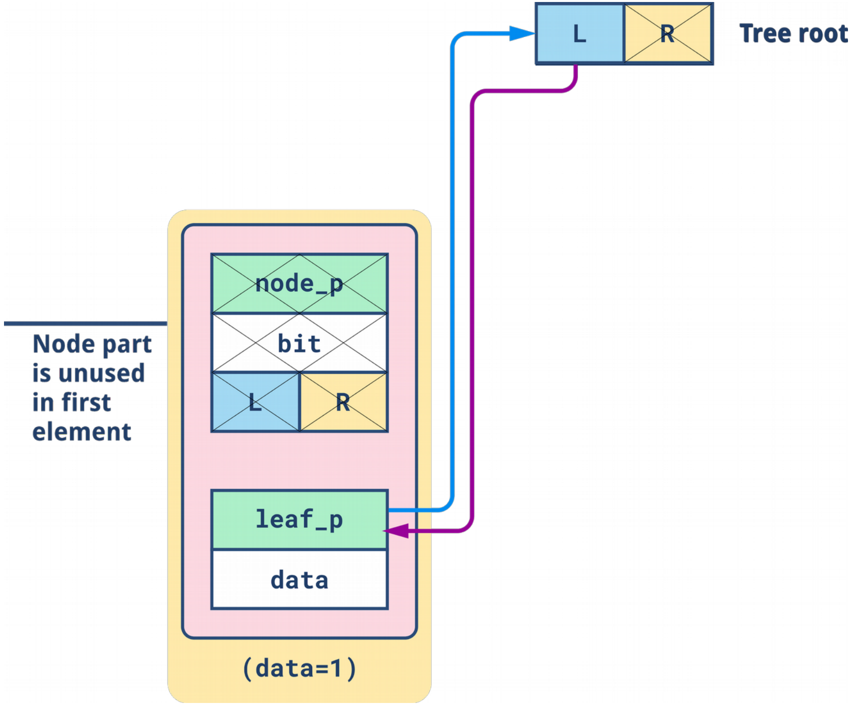
L

R

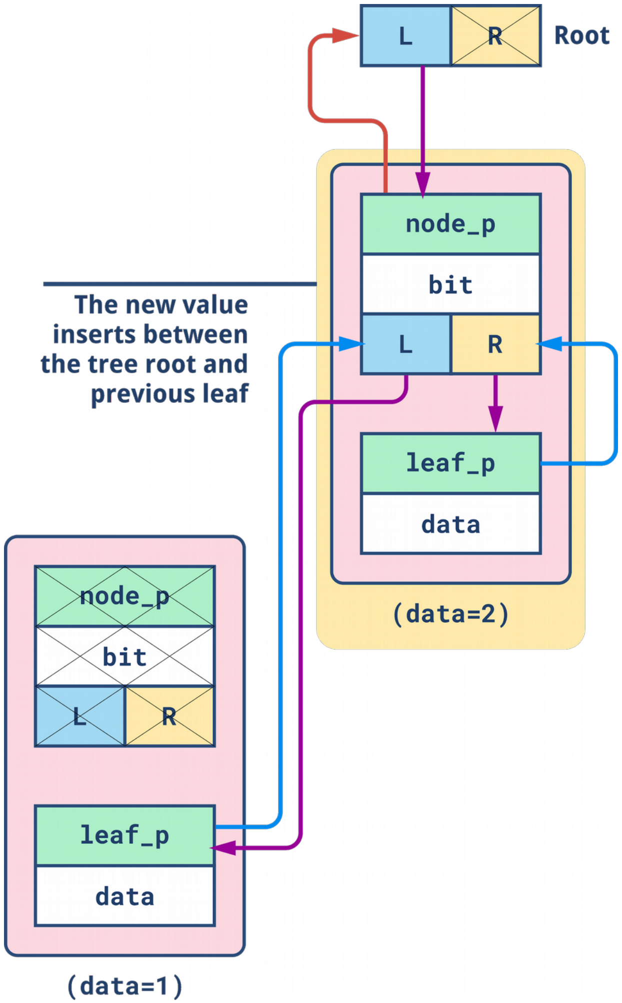
leaf_p

data

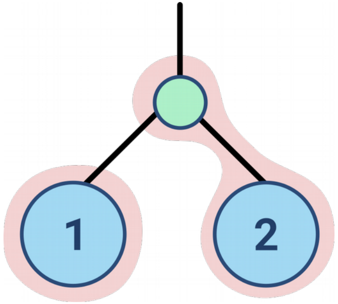

node_p
bit
L
R
node_p
bit
L
R
node_p
bit
L
R
node_p
bit
L
R
node_p
bit
L
R
leaf_p
dat
leaf_p
dat
leaf_p
dat
leaf_p
dat
leaf_p
dat

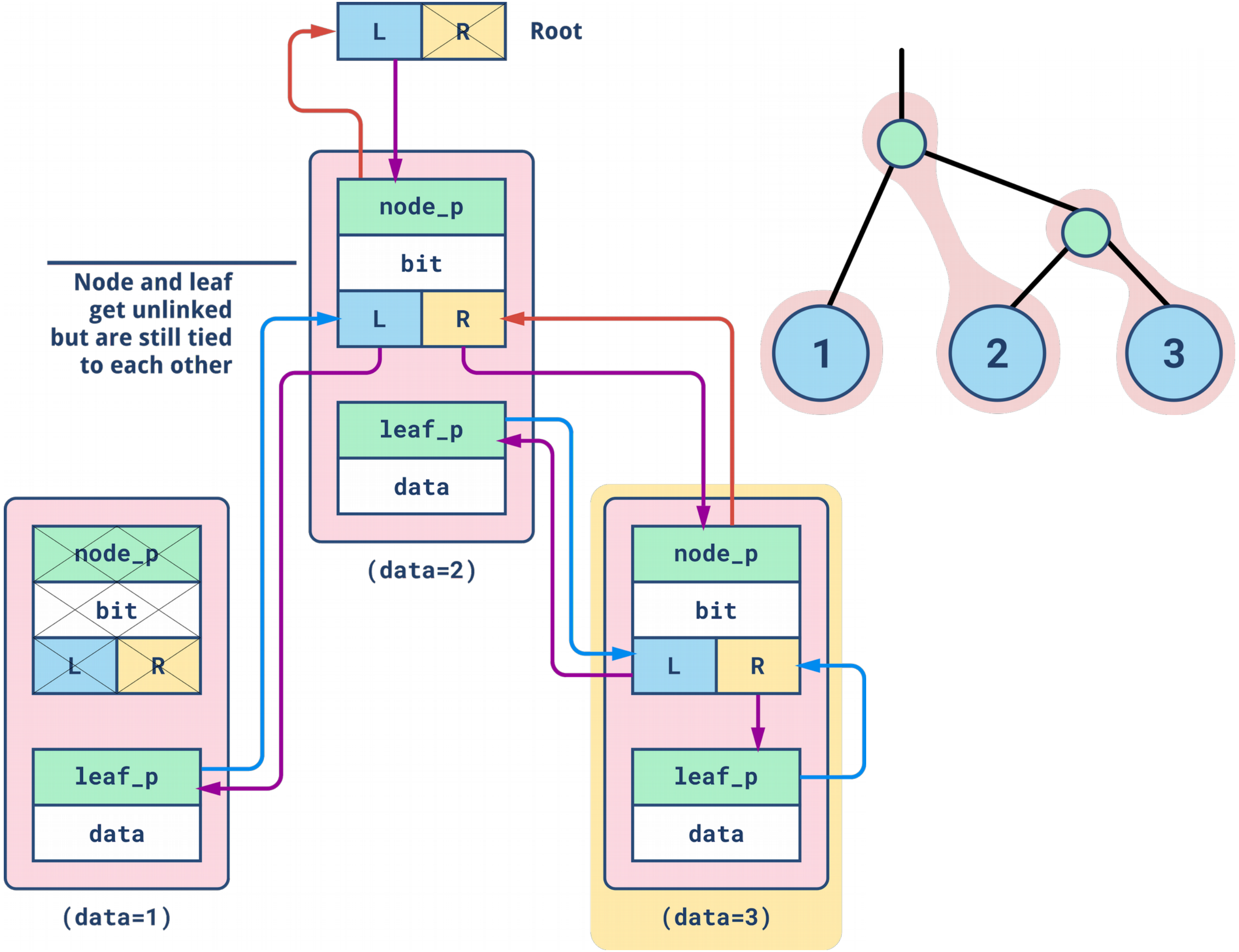


Inserting elements

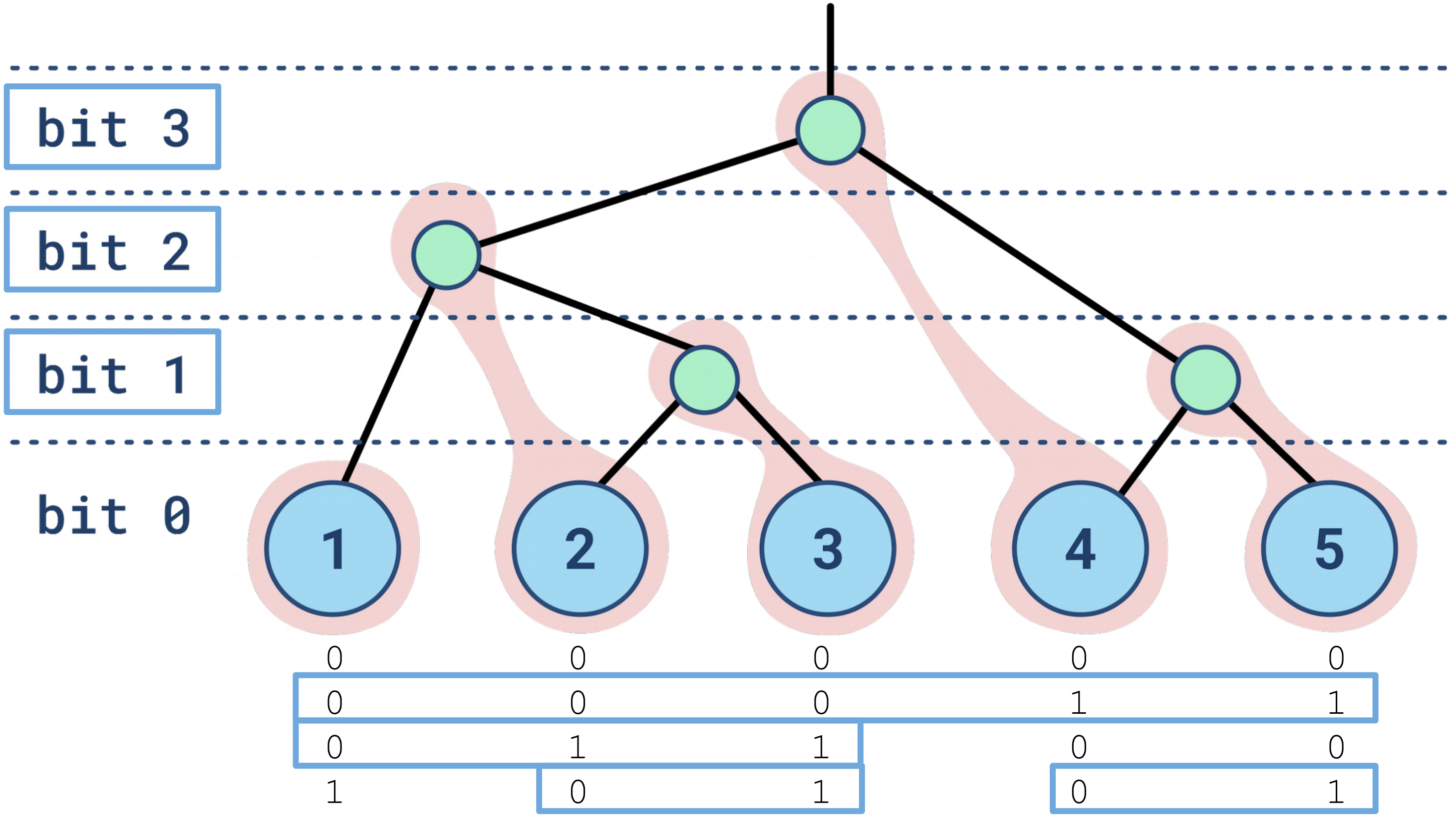


The new value inserts between the tree root and previous leaf



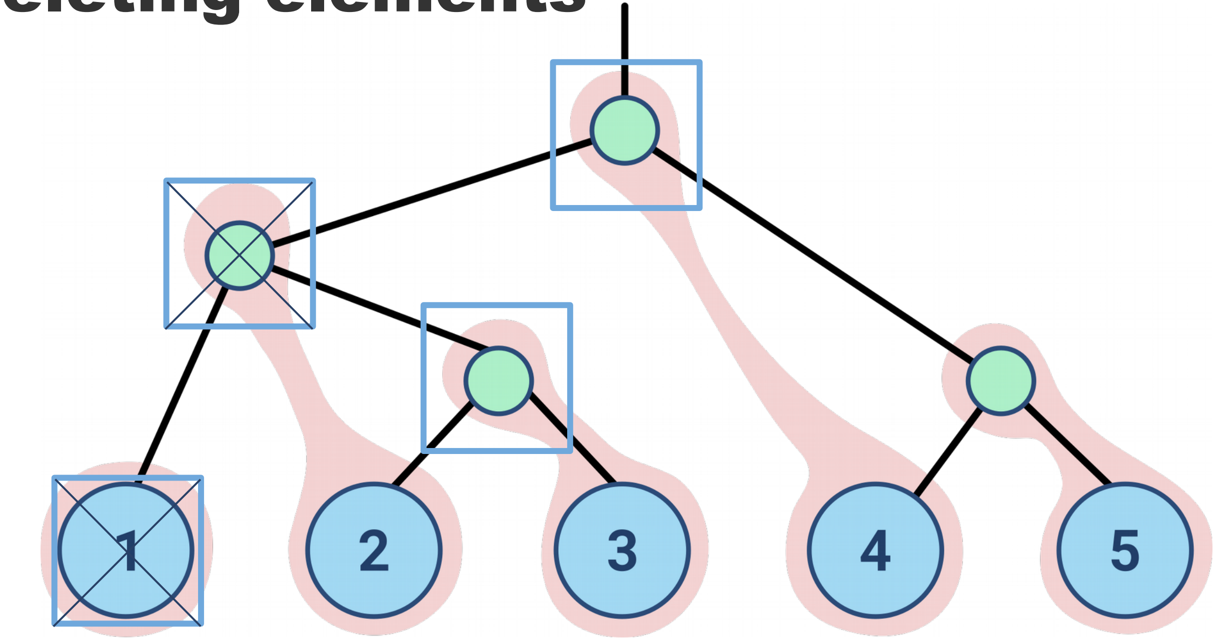


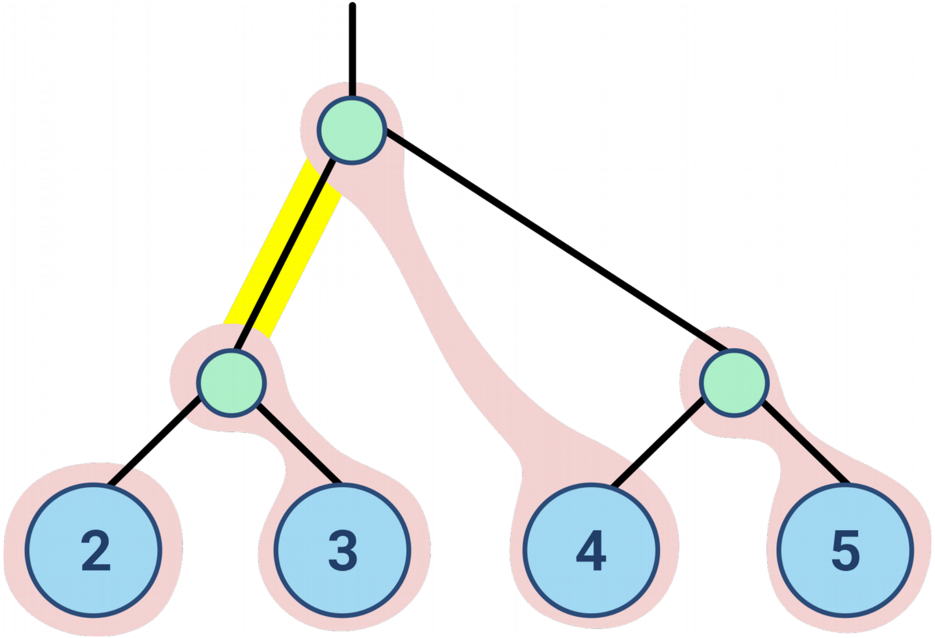
Node and leaf
get unlinked
but are still tied
to each other



0	0	0	0	0
0	0	0	1	1
0	1	1	0	0
1	0	1	0	1

Deleting elements




2
3
4
5

Implementation

CRAFTSMAN
NICKEL PLATED
CAYUGA

Pointer tagging

- 32 bits = 4 bytes word, 2 bits available
- 64 bits = 8 bytes word, 3 bits available

```
0x846010 = 100001000110000000010000
0x846030 = 100001000110000000110000
0x846050 = 100001000110000001010000
```

C is portable Assembler

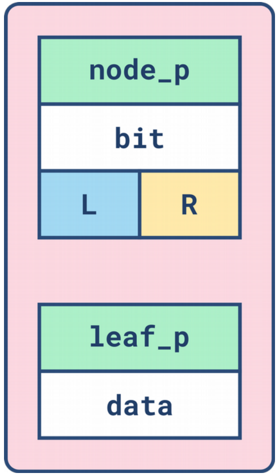
- regparm compiler directive (historical reason)
- forced inlining
- __builtin_expect
- ASM

Base structs

```
typedef void eb_troot_t;

struct eb_root {
      eb_troot_t    *b[2]; /* left and right branches */
};

struct eb_node {
      struct eb_root branches; /* branches, must be at the beginning */
      short int      bit;     /* link's bit position. */
      eb_troot_t    *node_p;  /* link node's parent */
      eb_troot_t    *leaf_p;  /* leaf node's parent */
};
```



ebtree/ebtree.h

Base functions

```
/* Return next leaf node after an existing leaf node, or NULL if none. */
static inline struct eb_node *eb_next(struct eb_node *node)
{
     eb_troot_t *t = node->leaf_p;
     while (eb_gettag(t) != EB_LEFT)
          /* Walking up from right branch, so we cannot be below root */
          t = (eb_root_to_node(eb_untag(t, EB_RGHT)))->node_p;

     /* Note that <t> cannot be NULL at this stage */
     t = (eb_untag(t, EB_LEFT))->b[EB_RGHT];
     if (eb_clrtag(t) == NULL)
          return NULL;
     return eb_walk_down(t, EB_LEFT);
}
```

eb_next() in ebtree/ebtree.h

EBtree data types

- eb32 / eb64
- ebpt for pointers
- ebim and ebis for indirect memory and strings
- ebmb and ebst for memory block and strings (allocated after just after the node)
- All support storage and ordered retrieval of duplicate keys

eb64 node

```
struct eb64_node {
        struct eb_node node; /* the tree node, must be at the beginning */
        u64 key;
};
```

ebtree/eb64tree.h

eb64 specific functions

```
troot = root->b[EB_LEFT];
if (unlikely(troot == NULL))
      return NULL;

while (1) {
      if ((eb_gettag(troot) == EB_LEAF)) {
            node = container_of(eb_untag(troot, EB_LEAF),
                                struct eb64_node, node.branches);
            if (node->key == x)
                  return node;
            else
                  return NULL;
      }
```

eb64_lookup() in ebtree/eb64tree.h

eb64 specific functions

```
            node = container_of(eb_untag(troot, EB_NODE),
                                struct eb64_node, node.branches);

            y = node->key ^ x;
            if (!y) {
                  /* Either we found the node which holds the key, or
                   * we have a dup tree. */
                  return node;
            }
            if ((y >> node->node.bit) >= EB_NODE_BRANCHES)  /* 2 */
                  return NULL; /* no more common bits */
            troot = node->node.branches.b[(x >> node->node.bit) &
                                          EB_NODE_BRANCH_MASK];
      }
}
```

eb64_lookup() in ebtree/eb64tree.h

Production use

HAProxy tasks

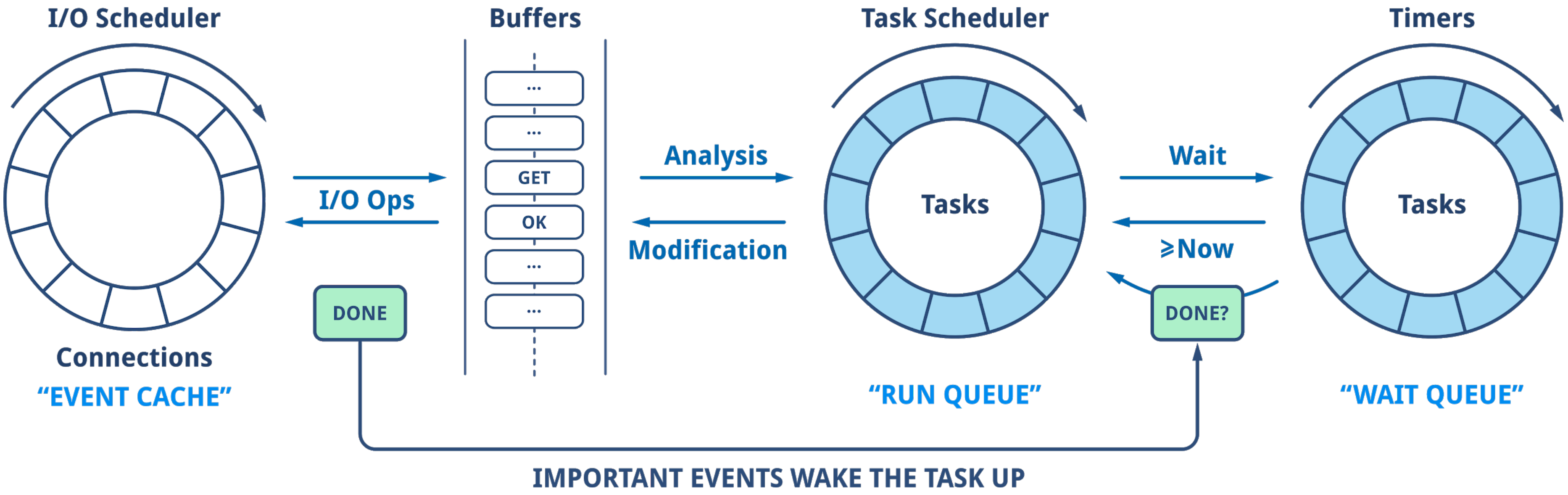
- computational load associated with a proxied connection
- active
- suspended
- millisecond resolution

Suspended HAProxy tasks

- EBtree
- indexed on expiration date

Active HAProxy tasks

- EBtree
- indexed on expiration date, taking priority into consideration


I/O Scheduler
Buffers
Task Scheduler
Timers
...
...
GET
OK
...
...
I/O Ops
Analysis
Modification
Tasks
Wait
≥Now
Tasks
DONE
DONE?
Connections
"EVENT CACHE"
"RUN QUEUE"
"WAIT QUEUE"
IMPORTANT EVENTS WAKE THE TASK UP

HAProxy event loop

```
while (1) {
        /* Process a few tasks */
        process_runnable_tasks();

        /* Check if we can expire some tasks */
        next = wake_expired_tasks();

        /* expire immediately if events are pending */
        if (fd_cache_num || run_queue) next = now_ms;

        /* The poller will ensure it returns around <next> */
        cur_poller.poll(&cur_poller, next);
        fd_process_cached_events();
}
```

run_poll_loop() in haproxy-1.7.x/src/haproxy.c

mension data
Deloitte
SAPINDA
BORA
etixx
QUICK-STEP
janom
LIDL
GIANT
Alpecin

Task struct

```
/* The base for all tasks */
struct task {
      struct eb32_node rq;          /* ebtree node used to hold the task in the run queue */
      struct eb32_node wq;          /* ebtree node used to hold the task in the wait queue */
      unsigned short state;         /* task state : bit field of TASK_* */
      short nice;             /* the task's current nice value from -1024 to +1024 */
      unsigned int calls;           /* number of times ->process() was called */
      struct task * (*process)(struct task *t);  /* the function which processes the task */
      void *context;                /* the task's context */
      int expire;             /* next expiration date for this task, in ticks */
};
```

haproxy-1.7.x/include/types/task.h

Scheduling tasks for later

```
      if (likely(last_timer && last_timer->node.bit < 0 &&
               last_timer->key == task->wq.key && last_timer->node.node_p)) {
            eb_insert_dup(&last_timer->node, &task->wq.node);
            if (task->wq.node.bit < last_timer->node.bit)
                  last_timer = &task->wq;
            return;
      }
      eb32_insert(&timers, &task->wq);

      /* Make sure we don't assign the last_timer to a node-less entry */
      if (task->wq.node.node_p && (!last_timer || (task->wq.node.bit < last_timer->node.bit)))
            last_timer = &task->wq;
      return;
}
```

__task_queue() in haproxy-1.7.x/src/task.c

Waking up tasks to run

```
        if (likely(t->nice)) {
                int offset;

                niced_tasks++;
                if (likely(t->nice > 0))
                        offset = (unsigned)((tasks_run_queue * (unsigned int)t->nice) / 32U);
                else
                        offset = -(unsigned)((tasks_run_queue * (unsigned int)-t->nice) / 32U);
                t->rq.key += offset;
        }

        eb32_insert(&rqueue, &t->rq);
        rq_next = NULL;
        return t;
}
```

__task_wakeup() in haproxy-1.7.x/src/task.c

Running tasks

```
while (max_processed--) {
        if (unlikely(!rq_next)) {
                rq_next = eb32_lookup_ge(&rqueue, rqueue_ticks - TIMER_LOOK_BACK);
                if (!rq_next) {
                        /* we might have reached the end of the tree, typically because
                         * <rqueue_ticks> is in the first half and we're first scanning
                         * the last half. Let's loop back to the beginning of the tree now.
                         */
                        rq_next = eb32_first(&rqueue);
                        if (!rq_next)
                                break;
                }
        }
```

process_runnable_tasks() in haproxy-1.7.x/src/task.c

Running tasks

```
      t = eb32_entry(rq_next, struct task, rq);
      rq_next = eb32_next(rq_next);
      __task_unlink_rq(t);

      t->state |= TASK_RUNNING;
      t->calls++;
      t = t->process(t);

      if (likely(t != NULL)) {
            t->state &= ~TASK_RUNNING;
            if (t->expire)
                  task_queue(t);
      }
}
```

process_runnable_tasks() in haproxy-1.7.x/src/task.c

EBtree in HAProxy

- timers
- schedulers
- ACL
- stick-tables (stats, counters)
- LRU cache

EBtree performing in HAProxy

- Down to 100ns inserts
- > 200k TCP conn/s
- > 350k HTTP req/s
- scheduler using up only 3-5% CPU
- Halog utility - up to 4 million log lines per second
- 450000 BGP routes table: >2 million lookups per second

Yiddish English
LEARNING GERMAN BOOK
Cassell's Dutch Dictionary
YIDDISH
WORDS ON FIRE
DUTCH
SWEDISH
NORWEGIAN
PERSIAN GRAMMAR
modern pers
DARI practical dictic
Gaelic
irish grammar
welsh diction
OXFORD
501 RUSSIAN VERBS
The BIG SILVER Book of RUSSIAN VERBS
Oxford Chinese
Chinese FOR DUMMIES
ENGLISH-JAPANESE DICTIONARY
WILLIAM SAFIRE
IDIOT'S Learning
THE STORY OF FRENCH
LAROUSSE
OXFORD
Langenscheidt Standard Dictionary French
THE OXFORD-DUDEN PICTORIAL FRENCH & ENGLISH DICTIONARY
BEGINNER'S DICTIONARY
STREET FRENCH
Webster's New World Italian Dictionary
THE BANTAM NEW COLLEGE ITALIAN & ENGLISH DICTIONARY
The BIG SILVER Book of RUSSIAN VERBS
Russian
LEARNING RUSSIAN
ELEMENTARY ARABIC GRAM
BULGARIAN-ENGLISH DICTI
SERBO-CROAT
Bosnian
SLOVENE
201 POLISH VERBS
FARSI (PERSI
CZECH
THAI & ENGLISH
KOREAN FOR BEGINNERS
LAO FOR BEGINNERS
Mandarin
Swahili
WORDFEST!
Children's Writer's WORD BOOK
thesaurus
IDIOT'S Learning Itali
CASSELL'S ITALIAN
LEARNING ITALI
italian vocabulary
Spanish
LAROUSSE
SPANISH DICTIONARY
Pocket Oxford Spanish
BEGINNING HITTI
201 HEBREW VERBS
A HISTORY OF THE HEBREW LANGUAGE
HEBREW IN AMERICA
Webster's New World Hebrew Dictionary
Gesenius' HEBREW GRAMMAR
Webster's New World Hebrew Dictionary
Hebrew for the Rest of Us
501 HEBREW VERBS
Prayerbook Hebrew the Easy Way
SPOKEN LEBANESE
swahili
Diné Bizaad: Speak, Read, Write Na

LRU cache structs

```
struct lru64_list {
      struct lru64_list *n;
      struct lru64_list *p;
};

struct lru64_head {
      struct lru64_list list;
      struct eb_root keys;
      struct lru64  *spare;
      int cache_size;
      int cache_usage;
};
```

ebtree/examples/lru.h

LRU cache structs

```
struct lru64 {
      struct eb64_node node;        /* indexing key, typically a hash64 */
      struct lru64_list lru;        /* LRU list */
      void *domain;                 /* who this data belongs to */
      unsigned long long revision;  /* data revision (to avoid use-after-free) */
      void *data;                   /* returned value, user decides how to use this */
};
```

ebtree/examples/lru.h

LRU cache get/store

```
struct lru64 *lru64_get(unsigned long long key, struct lru64_head *lru,
                  void *domain, unsigned long long revision)
{
      struct eb64_node *node;
      struct lru64 *elem;

      if (!lru->spare) {
            if (!lru->cache_size)
                  return NULL;
            lru->spare = malloc(sizeof(*lru->spare));
            if (!lru->spare)
                  return NULL;
            lru->spare->domain = NULL;
      }
```

lru64_get() in ebtree/examples/lru.c

LRU cache get/store

```
/* Lookup or insert */
lru->spare->node.key = key;
node = __eb64_insert(&lru->keys, &lru->spare->node);
elem = container_of(node, typeof(*elem), node);

if (elem != lru->spare) {
      /* Existing entry found, check validity then move it at the head of the LRU list. */
      return elem;
}
else {
      /* New entry inserted, initialize and move to the head of the
       * LRU list, and lock it until commit. */
      lru->cache_usage++;
      lru->spare = NULL; // used, need a new one next time
}
```

lru64_get() in ebtree/examples/lru.c

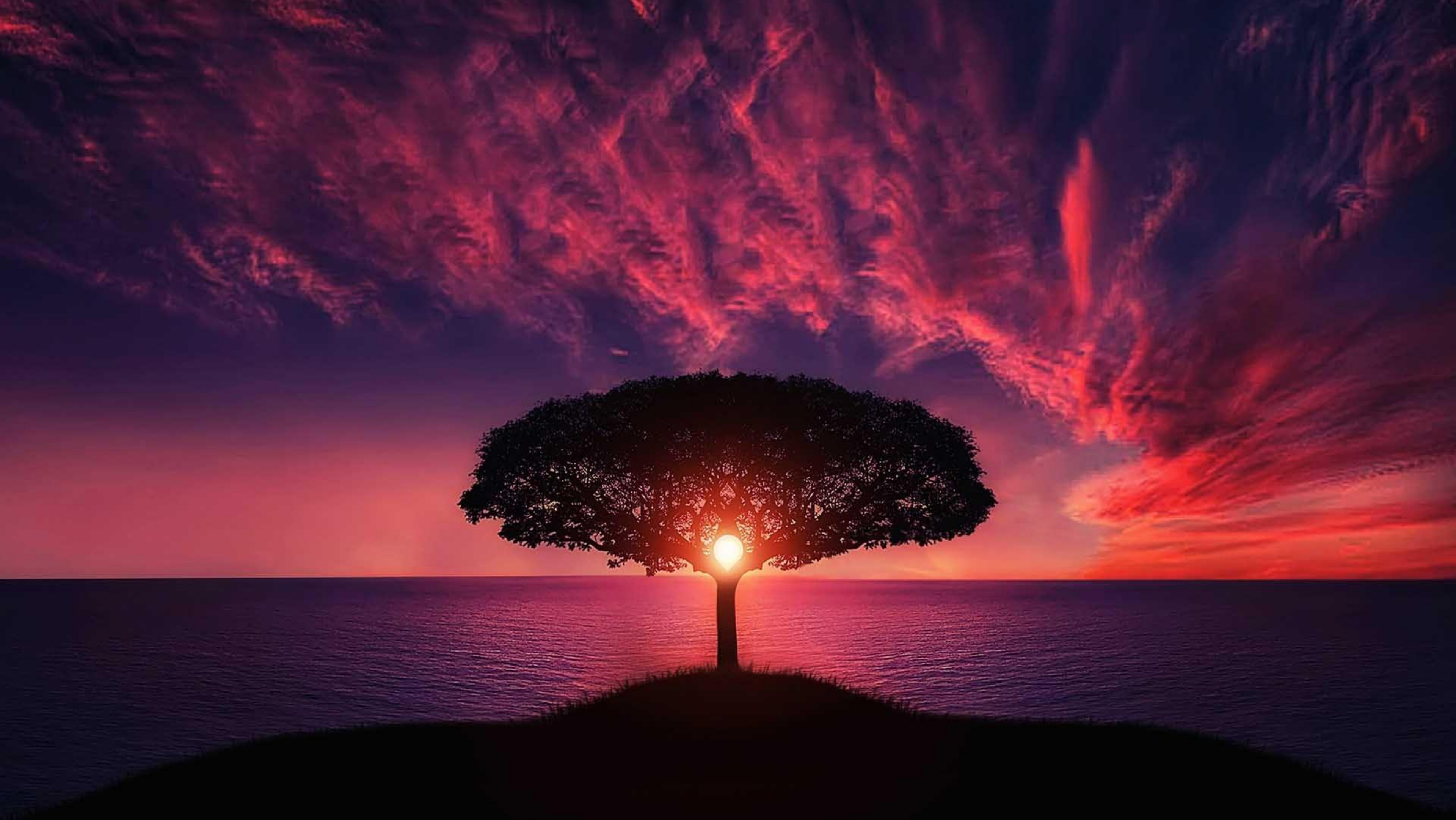
LRU cache get/store

```
if (lru->cache_usage > lru->cache_size) {
      struct lru64 *old;

      old = container_of(lru->list.p, typeof(*old), lru);
      if (old->domain) {
            /* not locked */
            LIST_DEL(&old->lru);
            __eb64_delete(&old->node);
            if (!lru->spare)
                  lru->spare = old;
            else
                  free(old);
            lru->cache_usage--;
      }
}
```

lru64_get() in ebtree/examples/lru.c

Results



EBtree features

- Fast tree descent & search
- Memory efficient
- Lookup by mask or prefix (i.e. IPv4 and IPv6)
- Optimized for inserts and deletes
- Great with bit-addressable data

Q&A

Check out EBtree at http://git.1wt.eu/web/ebtree.git/

Check out HAProxy at haproxy.org or haproxy.com

Join development at haproxy@formilux.org

aiharos@haproxy.com