PicoRuby Compiler

Monstarlab

Ruby Association Grant 2020 Report

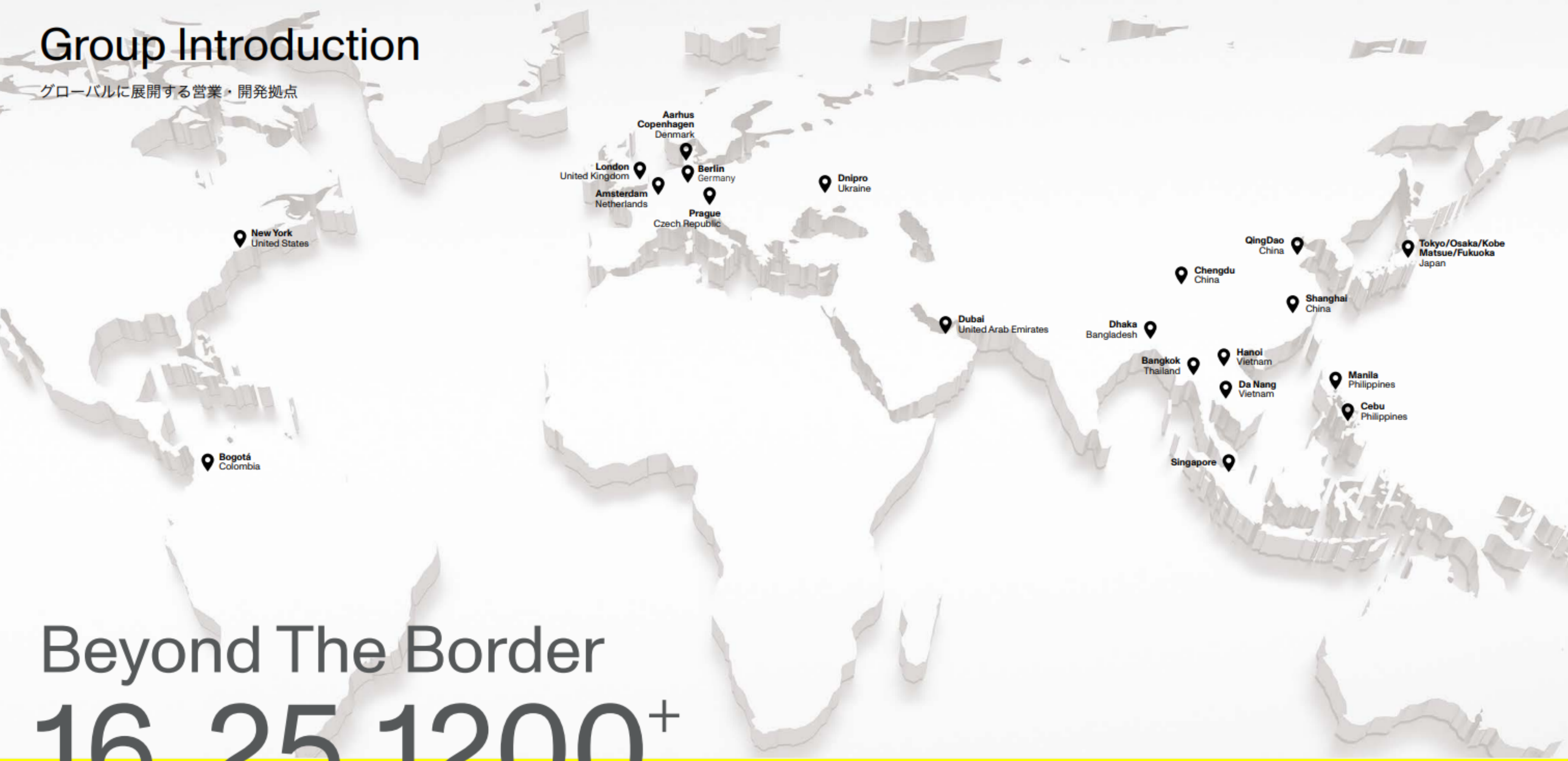
July 6, 2021

We are

- Monstarlab
- Established in 2006
- 85 Million USD of Capital (Nov. 2020)
- 1200+ Members in 25 Cities, 16 Countries
- Full-Time Remoties in Japan even before the pandemic
- Now Hiring! Reach out to me👦

Group Introduction

グローバルに展開する営業・開発拠点



Beyond The Border

16 Countries

25 Cities

1200+ Members

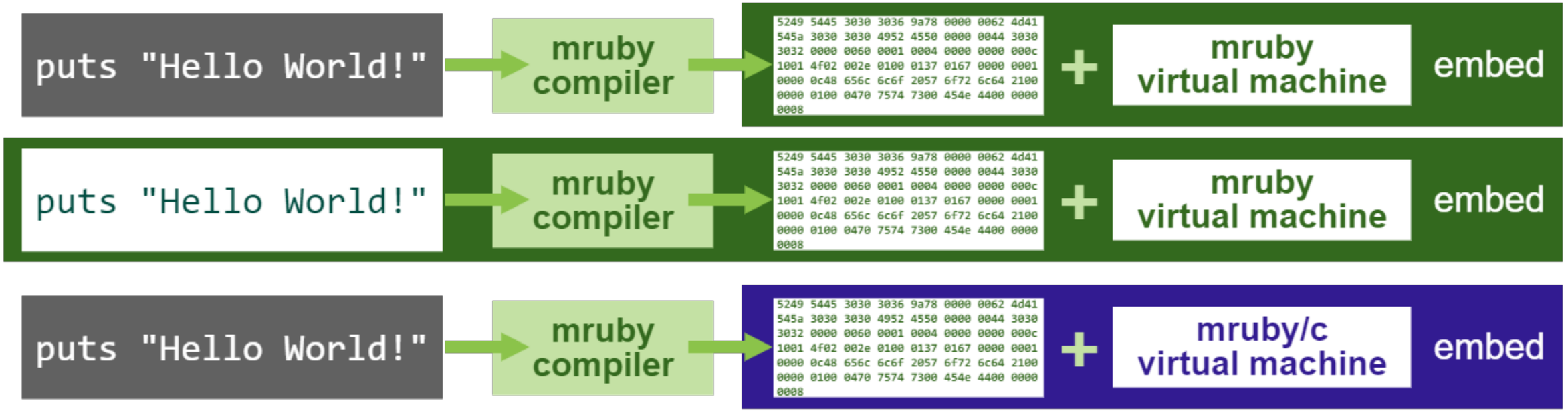
I am

- HASUMI Hitoshi
- Shimane development branch
- hasumikin@GitHub
- hasumikin@Twitter
  - 🙅hasumin（はすみん）
  - 🤦hasumikin（はすみきん）

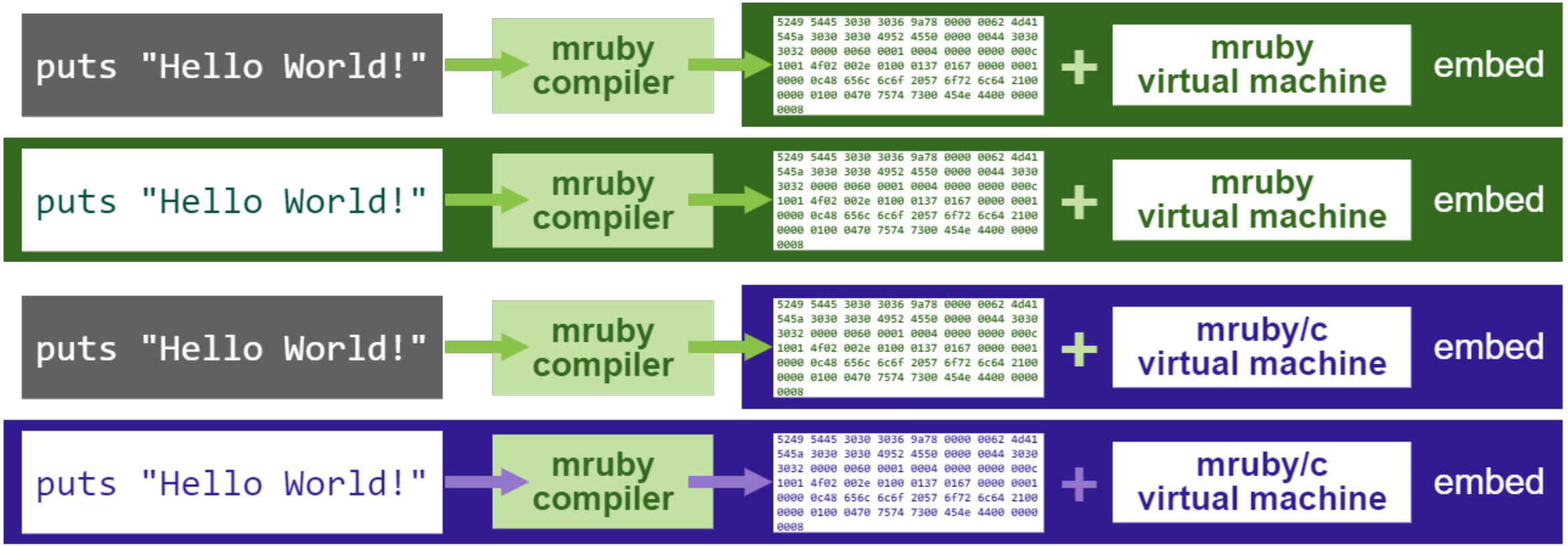
PicoRuby Compiler

Why? What?

Ruby for embedded system



Ruby for embedded system

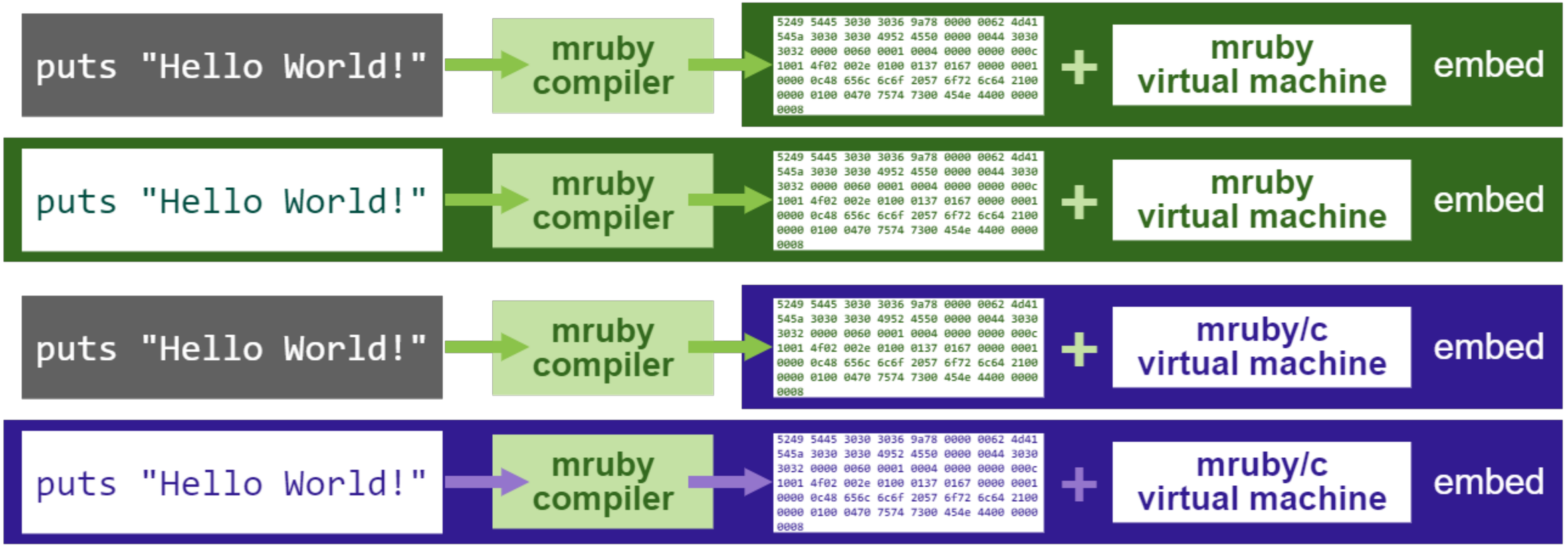


Doesn't make sense 🤔

Why doesn't it make sense?

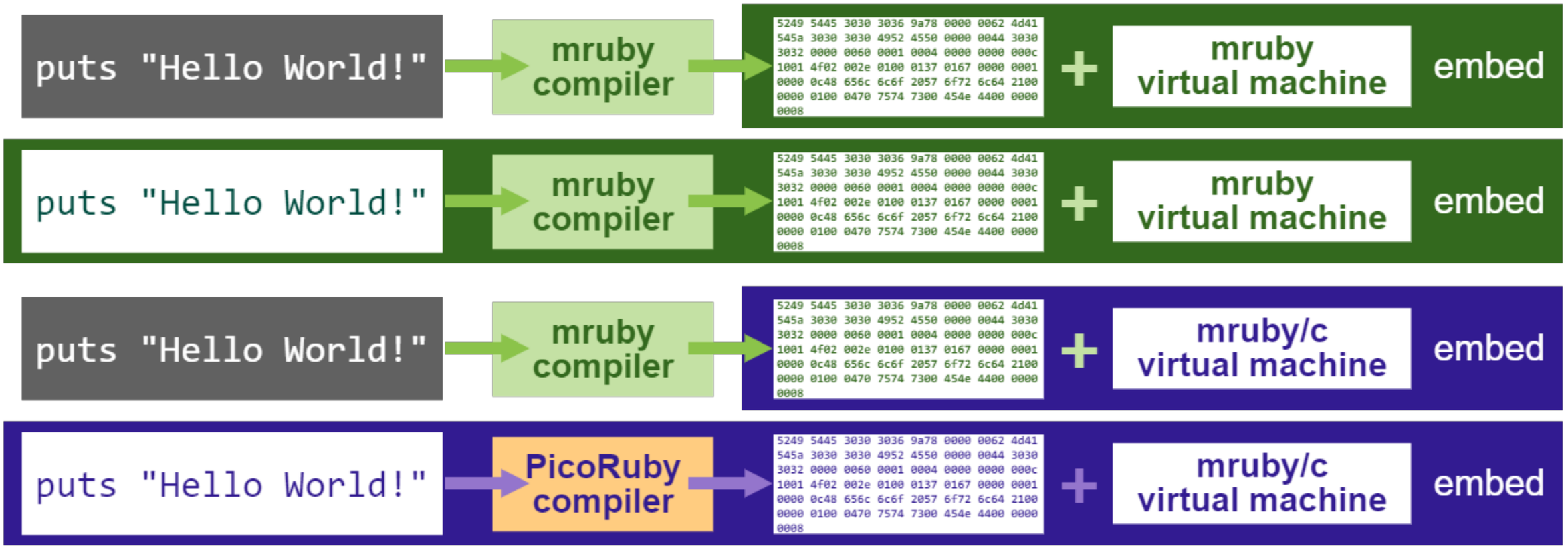
- mruby-compiler depends on mruby (mrb_state)
- You can use mruby-compiler with mruby/c VM
- However, the size of mruby-compiler spoils mruby/c's small footprint
- OK

Ruby for embedded system



Doesn't make sense 🤔

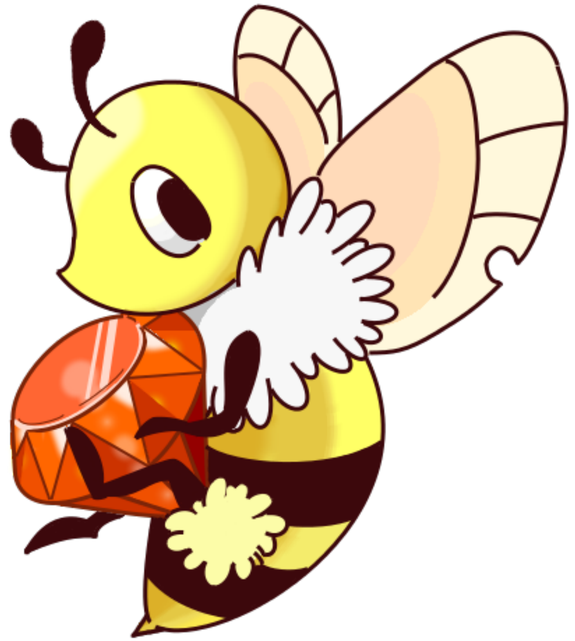
Ruby for embedded system



PicoRuby Compiler for one-chip microcontrollers 💪

Possibilities of PicoRuby Compiler

- REPL
  - Debugging on an actual device
  - Even with mruby VM
- One stop solution with mruby/c VM
  - For newbies, education, Smalruby etc.

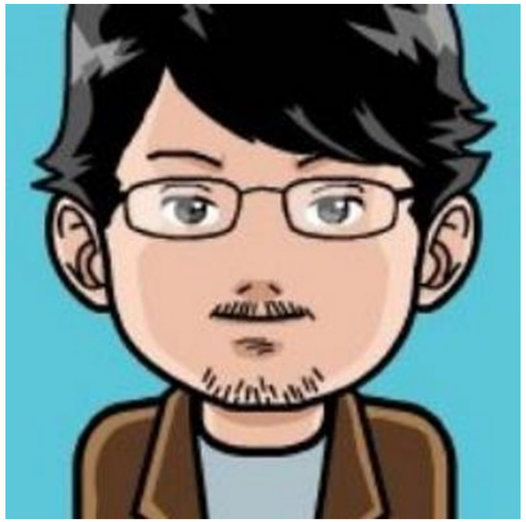


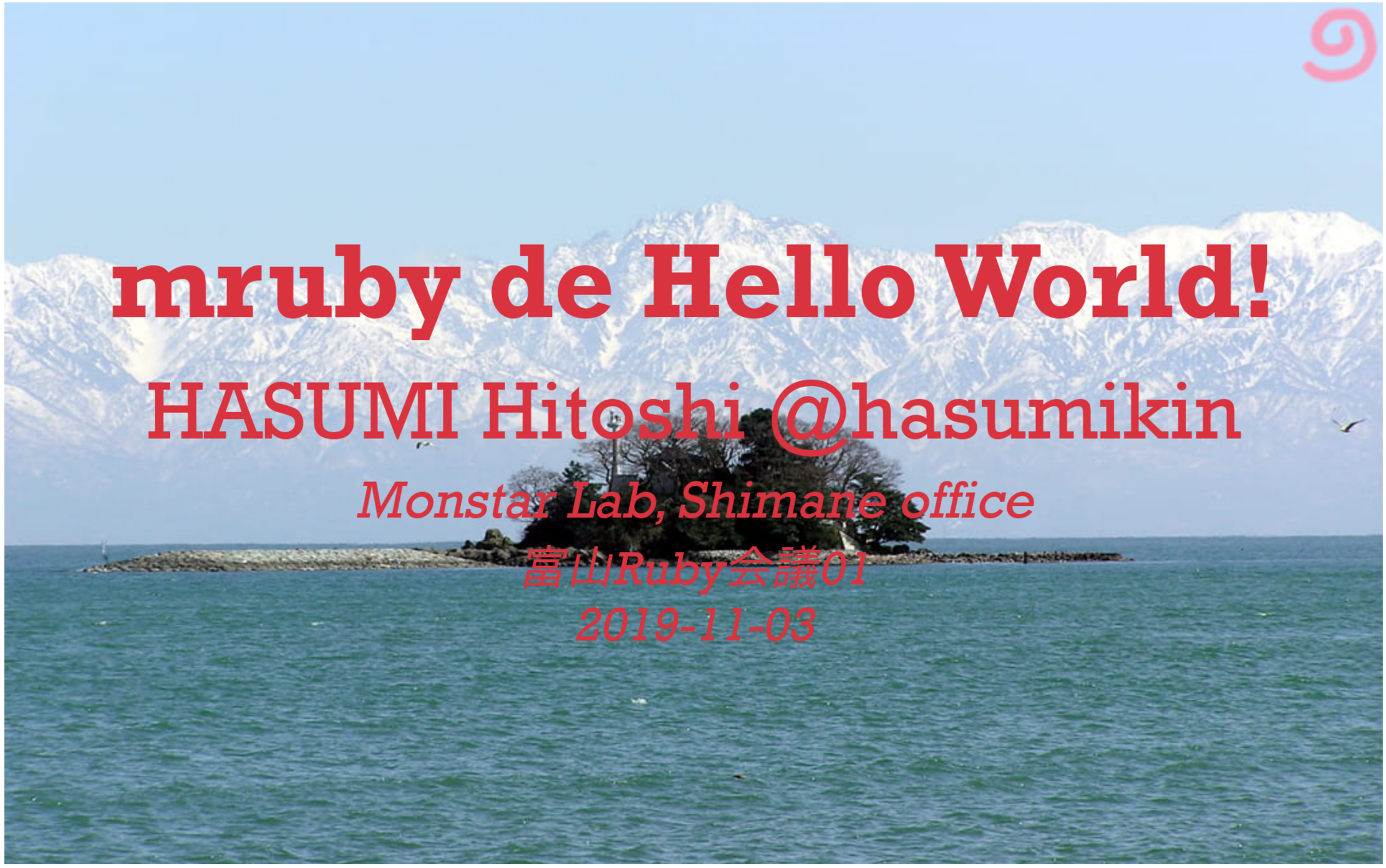
PicoRuby Compiler History

July 2019 -


IoT with mruby/c on the
Asahi-shuzo (Shimane)
HASUMI Hitoshi @hasumikin
Monstar Lab
Rubyビジネス創出展
July 19, 2018
KRUG

"Lemon would generate smaller binary than Bison."





mruby de Hello World!

HASUMI Hitoshi @hasumikin

Monstar Lab, Shimane office

富山Ruby会議01

2019-11-03

github.com/hasumikin/mmrbc.gem

- mini mruby compiler
- Tokenizer and Generator
  - CRuby
- Parser
  - C (Lemon) + ffi gem
- Able to compile only `puts "Hello World!"`
  - `[Identifier] "[String literal]"`

Jan 2020 -

Writing `mmruby` in C

then, `mv mmruby picoruby`

RA Grant Outcome

Goals

- Sufficient portion of the Ruby syntax to write IoT firmware
- Reduce RAM usage to execute on microcontrollers

Syntax - before RA Grant

```
# larger_script.rb
ary = Array.new(3)
ary[0] = {a: 123e2}
ary[0][:key] = "string"
ary[1] = %w(abc ABC Hello)
ary[2] = 0x1f
ary[3] = !true
puts "my name is #{self.class}, result is #{ary[2] * ary[0][:a]}"
p ary

=> my name is Object, result is 381300
   [{:a=>12300, :key=>"string"}, ["abc", "ABC", "Hello"], 31, false]
```

class, def, return, block, if/unless, case-when,
while/until, break, etc were unimplemented yet

Syntax - after RA Grant

```
$mutex = Mutex.new

$mutex.lock

class Float
  def modulo(right)
    left = self.to_f
    while left > right
      left -= right
    end
    left
  end
  def ceil_to_i
    n = self.to_i
    (self > n) ? (n + 1) : n
  end
end

class Keyboard
  GPIO_OUT = 1
  GPIO_IN  = 0

  HI = 1
  LO = 0

  MOD_KEYCODE = {
    KC_LCTL: 0b00000001,
    KC_LSFT: 0b00000010,
    KC_LALT: 0b00000100,
    KC_LGUI: 0b00001000,
    KC_RCTL: 0b00010000,
    KC_RSFT: 0b00100000,
    KC_RALT: 0b01000000,
    KC_RGUI: 0b10000000
  }

  # Due to PicoRuby's limitation,
  # a big array can't be created at once
  KEYCODE = [
    :KC_NO,                 # 0x00
    :KC_ROLL_OVER,
    :KC_POST_FAIL,
    :KC_UNDEFINED,
    :KC_A,
    :KC_B,
    :KC_C,
    :KC_D,
    :KC_E,
    :KC_F,
    :KC_G,
    :KC_H,
    :KC_I,
    :KC_J,
    :KC_K,
    :KC_L,
    :KC_M,                  # 0x10
    :KC_N,
    :KC_O,
    :KC_P,
    :KC_Q,
    :KC_R,
    :KC_S,
    :KC_T,
    :KC_U,
    :KC_V,
    :KC_W,
    :KC_X,
    :KC_Y,
    :KC_Z,
    :KC_1,
    :KC_2,
    :KC_3,                  # 0x20
    :KC_4,
    :KC_5,
    :KC_6,
    :KC_7,
    :KC_8,
    :KC_9,
    :KC_0,
    :KC_ENTER,
    :KC_ESCAPE,
    :KC_BSPACE,
    :KC_TAB,
    :KC_SPACE,
    :KC_MINUS,
    :KC_EQUAL,
    :KC_LBRACKET,
  ] + [
    :KC_RBRACKET,           # 0x30
    :KC_BSLASH,
    :KC_NONUS_HASH,
    :KC_SCOLON,
    :KC_QUOTE,
    :KC_GRAVE,
    :KC_COMMA,
    :KC_DOT,
    :KC_SLASH,
    :KC_CAPSLOCK,
    :KC_F1,
    :KC_F2,
    :KC_F3,
    :KC_F4,
    :KC_F5,
    :KC_F6,
    :KC_F7,                 # 0x40
    :KC_F8,
    :KC_F9,
    :KC_F10,
    :KC_F11,
    :KC_F12,
    :KC_PSCREEN,
    :KC_SCROLLLOCK,
    :KC_PAUSE,
    :KC_INSERT,
    :KC_HOME,
    :KC_PGUP,
    :KC_DELETE,
    :KC_END,
    :KC_PGDOWN,
    :KC_RIGHT,
    :KC_LEFT,               # 0x50
    :KC_DOWN,
    :KC_UP,
    :KC_NUMLOCK,
    :KC_KP_SLASH,
    :KC_KP_ASTERISK,
    :KC_KP_MINUS,
    :KC_KP_PLUS,
    :KC_KP_ENTER,
    :KC_KP_1,
    :KC_KP_2,
    :KC_KP_3,
    :KC_KP_4,
    :KC_KP_5,
    :KC_KP_6,
    :KC_KP_7,
  ] + [
    :KC_KP_8,               # 0x60
    :KC_KP_9,
    :KC_KP_0,
    :KC_KP_DOT,
    :KC_NONUS_BSLASH,
    :KC_APPLICATION,
    :KC_POWER,
    :KC_KP_EQUAL,
    :KC_F13,
    :KC_F14,
    :KC_F15,
    :KC_F16,
    :KC_F17,
    :KC_F18,
    :KC_F19,
    :KC_F20,
    :KC_F21,                # 0x70
    :KC_F22,
    :KC_F23,
    :KC_F24,
    :KC_EXECUTE,
    :KC_HELP,
    :KC_MENU,
    :KC_SELECT,
    :KC_STOP,
    :KC_AGAIN,
    :KC_UNDO,
    :KC_CUT,
    :KC_COPY,
    :KC_PASTE,
    :KC_FIND,
    :KC_MUTE,
    :KC_,                   # 0x80
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
    :KC_,
  ] + [
    :KC_LANG1,              # 0x90
    :KC_LANG2,
    :KC_LANG3,
    :KC_LANG4,
    :KC_LANG5,
    :KC_LANG6,
    :KC_LANG7,
    :KC_LANG8,
    :KC_LANG9,
    :KC_ALT_ERASE,
    :KC_SYSREQ,
    :KC_CANCEL,
    :KC_CLEAR,
    :KC_PRIOR,
    :KC_RETURN,
    :KC_SEPARATOR,
    :KC_OUT,                # 0xA0
    :KC_OPER,
    :KC_CLEAR_AGAIN,
    :KC_CRSEL,
    :KC_EXSEL,
  ]

  # Keycodes with SHIFT modifier
  KEYCODE_SFT = {
    KC_EXLM:            0x1e,
    KC_AT:              0x1f,
    KC_HASH:            0x20,
    KC_DLR:             0x21,
    KC_PERC:            0x22,
    KC_CIRC:            0x23,
    KC_AMPR:            0x24,
    KC_ASTER:           0x25,
    KC_LPRN:            0x26,
    KC_RPRN:            0x27,
    KC_UNDS:            0x2d,
    KC_PLUS:            0x2e,
    KC_LCBR:            0x2f,
    KC_RCBR:            0x30,
    KC_PIPE:            0x31,
#   KC_TILD:            0x32,
    KC_COLON:           0x33,
    KC_DQUO:            0x34,
    KC_TILD:            0x35,
    KC_LABK:            0x36,
    KC_RABK:            0x37,
    KC_QUES:            0x38,
  }

  def initialize
    @before_filters = Array.new
    @keymaps = Hash.new
    @mode_keys = Array.new
    @switches = Array.new
    @layer_names = Array.new
    @layer = :default
    @split = false
    @anchor = true
    @anchor_left = true # so-called "master left"
    @uart_pin = 1
    $rgb = nil
  end

  attr_accessor :split, :uart_pin
  attr_reader :layer

  # TODO: OLED, SDCard
  def append(feature)
    case feature.class
    when RGB
      $rgb = feature
    end
  end

  def start_rgb
    $rgb.ws2812_resume if $rgb
  end

  # val should be treated as `:left` if it's anything other than `:right`
  def set_anchor(val)
    @anchor_left = false if val == :right
  end

  def init_pins(rows, cols)
    if @split
      sleep 2 # Wait until USB ready
      @anchor = tud_mounted?
      if @anchor
        uart_rx_init(@uart_pin)
      else
        uart_tx_init(@uart_pin)
      end
    end
    @rows = rows
    @cols = cols
    @rows.each do |pin|
      gpio_init(pin)
      gpio_set_dir(pin, GPIO_OUT);
      gpio_put(pin, HI);
    end
    @cols.each do |pin|
      gpio_init(pin)
      gpio_set_dir(pin, GPIO_IN);
      gpio_pull_up(pin);
    end
    # for split type
    @offset_a = (@cols.size / 2.0).ceil_to_i
    @offset_b = @cols.size * 2 - @offset_a - 1
  end

  # Input
  #   name:    default, map: [ [ :KC_A, :KC_B, :KC_LCTL, :MACRO_1 ],... ]
  # ↓
  # Result
  #   layer: { default:      [ [ -0x04, -0x05, 0b00000001, :MACRO_1 ],... ] }
  def add_layer(name, map)
    new_map = Array.new(@rows.size)
    row_index = 0
    col_index = 0
    cols_size = @split ? @cols.size * 2 : @cols.size
    map.each do |key|
      new_map[row_index] = Array.new(@cols.size) if col_index == 0
      keycode_index = KEYCODE.index(key)
      new_map[row_index][col_index] = if keycode_index
        keycode_index * -1
      elsif KEYCODE_SFT[key]
        (KEYCODE_SFT[key] + 0x100) * -1
      elsif MOD_KEYCODE[key]
        MOD_KEYCODE[key]
      else
        key
      end
      if col_index == cols_size - 1
        col_index = 0
        row_index += 1
      else
        col_index += 1
      end
    end
    @keymaps[name] = new_map
    @layer_names << name
  end

  # param[0] :on_release
  # param[1] :on_hold
  # param[2] :release_threshold
  # param[3] :repush_threshold
  def define_mode_key(key_name, param)
    on_release = param[0]
    on_hold = param[1]
    release_threshold = param[2]
    repush_threshold = param[3]
    @keymaps.each do |layer, map|
      map.each_with_index do |row, row_index|
        row.each_with_index do |key_symbol, col_index|
          if key_name == key_symbol
            # @type var on_release_action: Integer | Array[Integer] | Proc | nil
            on_release_action = case on_release.class
            when Symbol
              # @type var on_release: Symbol
              keycode_index = KEYCODE.index(on_release)
              if keycode_index
                keycode_index * -1
              elsif KEYCODE_SFT[on_release]
                (KEYCODE_SFT[on_release] + 0x100) * -1
              end
            when Array
              # @type var on_release: Array[Symbol]
              # @type var ary: Array[Integer]
              ary = Array.new
              on_release.each do |sym|
                keycode_index = KEYCODE.index(sym)
                ary << if keycode_index
                  keycode_index * -1
                elsif KEYCODE_SFT[sym]
                  (KEYCODE_SFT[sym] + 0x100) * -1
                else # Should be a modifier
                  MOD_KEYCODE[sym]
                end
              end
              ary
            when Proc
              # @type var on_release: Proc
              on_release
            end
            on_hold_action = if on_hold.is_a?(Symbol)
              # @type var on_hold: Symbol
              MOD_KEYCODE[on_hold] ? MOD_KEYCODE[on_hold] : on_hold
            else
              # @type var on_hold: Proc
              on_hold
            end
            @mode_keys << {
              layer:           layer,
              on_release:      on_release_action,
              on_hold:         on_hold_action,
              release_threshold: (release_threshold || 0),
              repush_threshold:  (repush_threshold || 0),
              switch:          [row_index, col_index],
              prev_state:      :released,
              pushed_at:       0,
              released_at:     0,
            }
          end
        end
      end
    end
  end

  # MOD_KEYCODE = {
  #   KC_LCTL: 0b00000001,
  #   KC_LSFT: 0b00000010,
  #   KC_LALT: 0b00000100,
  #   KC_LGUI: 0b00001000,
  #   KC_RCTL: 0b00010000,
  #   KC_RSFT: 0b00100000,
  #   KC_RALT: 0b01000000,
  #   KC_RGUI: 0b10000000
  # }
  def invert_sft
    if ( (@modifier & 0b00000010) | (@modifier & 0b00100000) ) > 0
      @modifier &= 0b11011101
    else
      @modifier |= 0b00000010
    end
  end

  def before_report(&block)
    @before_filters << block
  end

  def keys_include?(key)
    keycode = KEYCODE.index(key)
    !keycode.nil? && @keycodes.include?(keycode.chr)
  end

  def action_on_release(mode_key)
    case mode_key.class
    when Fixnum
      # @type var mode_key: Integer
      if mode_key < -255
        @keycodes << ((mode_key + 0x100) * -1).chr
        @modifier |= 0b00000010
      else
        @keycodes << (mode_key * -1).chr
      end
    when Array
      # @type var mode_key: Array[Integer]
      0 # `steep check` will fail if you remove this line 🤔
      mode_key.each do |key|
        if key < -255
          @keycodes << ((key + 0x100) * -1).chr
          @modifier |= 0b00000010
        elsif key < 0
          @keycodes << (key * -1).chr
        else # Should be a modifier
          @modifier |= key
        end
      end
    when Proc
      # @type var mode_key: Proc
      mode_key.call
    end
  end

  def action_on_hold(mode_key)
    case mode_key.class
    when Fixnum
      # @type var mode_key: Integer
      @modifier |= mode_key
    when Symbol
      # @type var mode_key: Symbol
      @layer = mode_key
    when Proc
      # @type var mode_key: Proc
      mode_key.call
    end
  end

  #
  ##############################################################
  #  For those who are willing to contribute to PRK Firmware:
  #
  #  The author has intentionally made this method big and
  #  redundant for resilience against a change of spec.
  #  Please refrain from "refactoring" for a while.
  #
  ##############################################################
  def start!
    start_rgb if $rgb
    @keycodes = Array.new
    # To avoid unintentional report on startup
    # which happens only on Sparkfun Pro Micro RP2040
    if @split
      sleep_ms 100
      while true
        data = uart_getc
        break if data.nil?
      end
    end
    while true
      now = board_millis
      @keycodes.clear

      @switches.clear
      @modifier = 0

      # detect physical switches that are pushed
      @rows.each_with_index do |row_pin, row|
        gpio_put(row_pin, LO)
        @cols.each_with_index do |col_pin, col|
          if gpio_get(col_pin) == LO
            col_data = if @anchor_left
                         if @anchor
                           # left
                           col
                         else
                           # right
                           (col - @offset_a) * -1 + @offset_b
                         end
                       else # right side is the anchor
                         unless @anchor
                           # left
                           col
                         else
                           # right
                           (col - @offset_a) * -1 + @offset_b
                         end
                       end
            @switches << [row, col_data]
          end
          break if @switches.size >= @cols.size
        end
        gpio_put(row_pin, HI)
      end

      # TODO: more features
      $rgb.fifo_push(true) if $rgb && !@switches.empty?

      # Receive switches from partner
      if @split && @anchor
        sleep_ms 5
        while true
          data = uart_getc
          break if data == nil
          # @type var data: Integer
          switch = [data >> 5, data & 0b00011111]
          # To avoid chattering
          @switches << switch unless @switches.include?(switch)
        end
      end

      if @anchor
        desired_layer = @layer
        @mode_keys.each do |mode_key|
          next if mode_key[:layer] != @layer
          if @switches.include?(mode_key[:switch])
            case mode_key[:prev_state]
            when :released
              mode_key[:pushed_at] = now
              mode_key[:prev_state] = :pushed
              on_hold = mode_key[:on_hold]
              if on_hold.is_a?(Symbol) && @layer_names.index(desired_layer).to_i < @layer_names.index(on_hold).to_i
                desired_layer = on_hold
              end
            when :pushed
              if !mode_key[:on_hold].is_a?(Symbol) && (now - mode_key[:pushed_at]) > mode_key[:release_threshold])
                action_on_hold(mode_key[:on_hold])
              end
            when :pushed_then_released
              if now - mode_key[:released_at] <= mode_key[:repush_threshold]
                mode_key[:prev_state] = :pushed_then_released_then_pushed
              end
            when :pushed_then_released_then_pushed
              action_on_release(mode_key[:on_release])
            end
          else
            case mode_key[:prev_state]
            when :pushed
              if now - mode_key[:pushed_at] <= mode_key[:release_threshold]
                action_on_release(mode_key[:on_release])
                mode_key[:prev_state] = :pushed_then_released
              else
                mode_key[:prev_state] = :released
              end
              mode_key[:released_at] = now
              @layer = @prev_layer || :default
              @prev_layer = nil
            when :pushed_then_released
              if now - mode_key[:released_at] > mode_key[:release_threshold]
                mode_key[:prev_state] = :released
              end
            when :pushed_then_released_then_pushed
              mode_key[:prev_state] = :released
            end
          end
        end
      end

      if @layer != desired_layer
        @prev_layer ||= @layer
        action_on_hold(desired_layer)
      end

      keymap = @keymaps[@locked_layer || @layer]
      @switches.each do |switch|
        keycode = keymap[switch[0]][switch[1]]
        next unless keycode.is_a?(Fixnum)
        if keycode < -255 # Key with SHIFT
          @keycodes << ((keycode + 0x100) * -1).chr
          @modifier |= 0b00100000
        elsif keycode < 0 # Normal keys
          @keycodes << (keycode * -1).chr
        else # Modifier keys
          @modifier |= keycode
        end
      end

      (6 - @keycodes.size).times do
        @keycodes << "\#000"
      end

      @before_filters.each do |block|
        block.call
      end

      report_hid(@modifier, @keycodes.join)

      if @switches.empty? && @locked_layer.nil?
        @layer = :default
      elsif @locked_layer
        # @type ivar @locked_layer: Symbol
        @layer = @locked_layer
      end
    else
      @switches.each do |switch|
        # 0b11111111
        #   ^^^       row number (0 to 7)
        #      ^^^^^  col number (0 to 31)
        uart_puts_raw((switch[0] << 5) + switch[1])
      end

      time = 10 - (board_millis - now)
      sleep_ms(time) if time > 0
    end

  end

  #
  # Actions can be used in keymap.rb
  #

  # Raises layer and keeps it
  def raise_layer
    current_index = @layer_names.index(@locked_layer || @layer)
    return if current_index.nil?
    if current_index < @layer_names.size - 1
      # @type var current_index: Integer
      @locked_layer = @layer_names[current_index + 1]
    else
      @locked_layer = @layer_names.first
    end
  end

  # Lowers layer and keeps it
  def lower_layer
    current_index = @layer_names.index(@locked_layer || @layer)
    return if current_index.nil?
    if current_index == 0
      @locked_layer = @layer_names.last
    else
      # @type var current_index: Integer
      @locked_layer = @layer_names[current_index - 1]
    end
  end

  # Switch to specified layer
  def lock_layer(layer)
    @locked_layer = layer
  end

  def unlock_layer
    @locked_layer = nil
  end

end

$mutex.unlock
```

RAM consumption

```
$ valgrind                          \
    --tool=massif                   \
    --stacks=yes                    \
    path/to/(mrbc¦picorbc)          \
    source.rb

$ ms_print massif.out.xxx ¦ less
```

RAM consumption

RAM consumption

```
#                          before Grant                  after Grant
                        mrbc       picorbc       mrbc(3.0.0)  picorbc
=============================================================================
puts "Hello World!"  157.8 KB   11.17 KB      133.6 KB      15.43 KB
-----------------------------------------------------------------------------
larger_script.rb     162.9 KB   53.20 KB      135.2 KB      24.21 KB
```

- Note:
  - using glibc and Linux file system
  - on 64-bit architecture
    - All figures should possibly be about 35% smaller on 32-bit

RAM consumption

#	before Grant		after Grant	
	mrbc	picorbc	mrbc(3.0.0)	picorbc
puts "Hello World!"	157.8 KB	11.17 KB	133.6 KB	15.43 KB
larger_script.rb	162.9 KB	53.20 KB	135.2 KB	24.21 KB

- picorbc made a great progress on larger_script.rb 🎉
  - 53.20 KB -> 24.21 KB
- but got worse on `puts "Hello World!"`
  - 11.17 KB -> 15.43 KB ...🤔What happened?

PicoRuby Compiler

What happened?

What happened?

- 1. Reusing literal data as much as possible
- 2. Considering paddings & Pooled allocation
- 3. Freeing in loop instead of recursion

Reusing literal data as much as possible

- There were many wasteful, duplicated memory allocations of literal data
  - eg) Reallocates memory when Tokenizer gives data to Parser
- Refactored the code to keep data common in a compilation cycle as much as possible
- eg) Token data puts on memory should be reused as it is until exactly before generating VM code

Reusing literal data as much as possible

```
array[0] = :data
```

- Tokens
  - `array`, `[`, `0`, `]`, `=`, `:`, `data`
- Method name should be `[]=`, but where? how?
- Thus, some literal data should be exceptional
  - Kind of a strategic decision
  - Beware of memory leaks

Padding in a struct

```
struct LinkedList {
  struct LinkedList *next;  // 4 bytes (in 32-bit architecture)
  uint8_t           value;  // 1 byte
}

sizeof(LinkedList);
=> 5 🤦
=> 8 🤦
```

✔Data structure alignment (at least in C99)

pointer[4] + uint8_t[1] + **padding[3]** = sum[8]

👉You need to pack them well

Pooled allocation

```
LinkedList *top;
(...)
top->next->next->next->next; // 5 items in the list
```

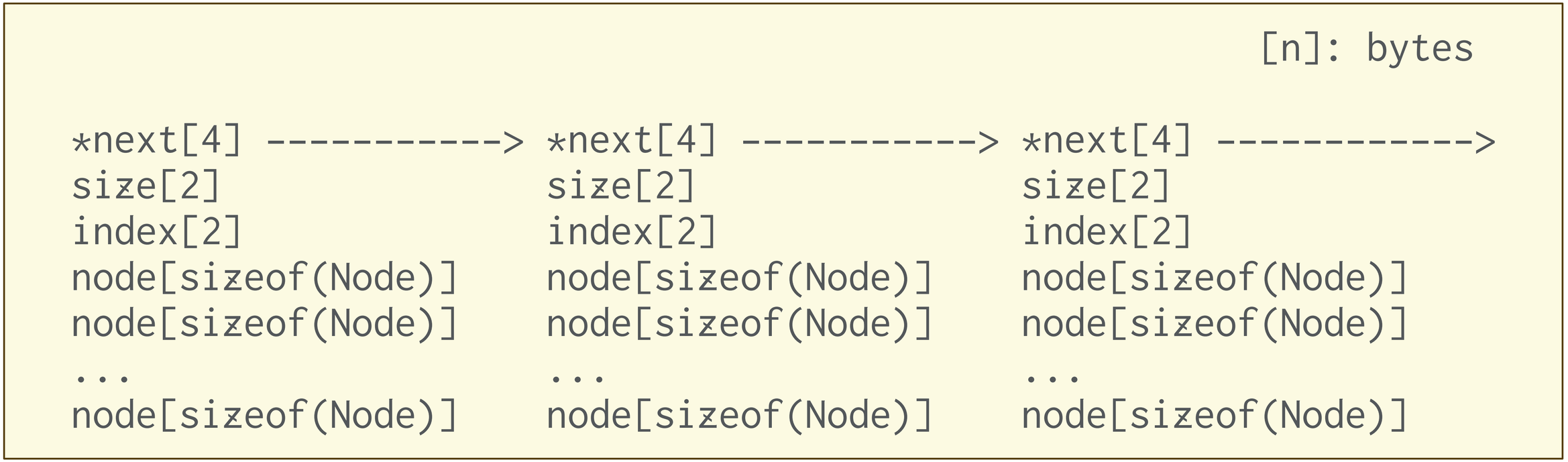
- Consumes 40 bytes to store 5 values of uint8_t
- Total size of pointers will be 1 KB if there are 250 items in a list

Pooled allocation 💡

Pooled allocation

```
typedef struct node_pool {
  NodePool *next;
  uint16_t size;
  uint16_t index;
  Node *nodes;
} NodePool;

NodePool *newNodePool() {
  size_t size = sizeof(NodePool) + sizeof(Node) * POOL_SIZE;
  NodePool *node_pool = (NodePool *)alloc(size);
  memset(node_pool, 0, size);
  node_pool->next = NULL;
  node_pool->size = POOL_SIZE;
  node_pool->index = 0;
  return node_pool;
}

/* node_pool.nodes[index] */
(Node *)((Node *)(&node_pool->nodes) + node_pool->index);
```

Pooled allocation



- Reduces paddings of data structure alignment
- Reduces the number of pointers
- Reduces fragmentation

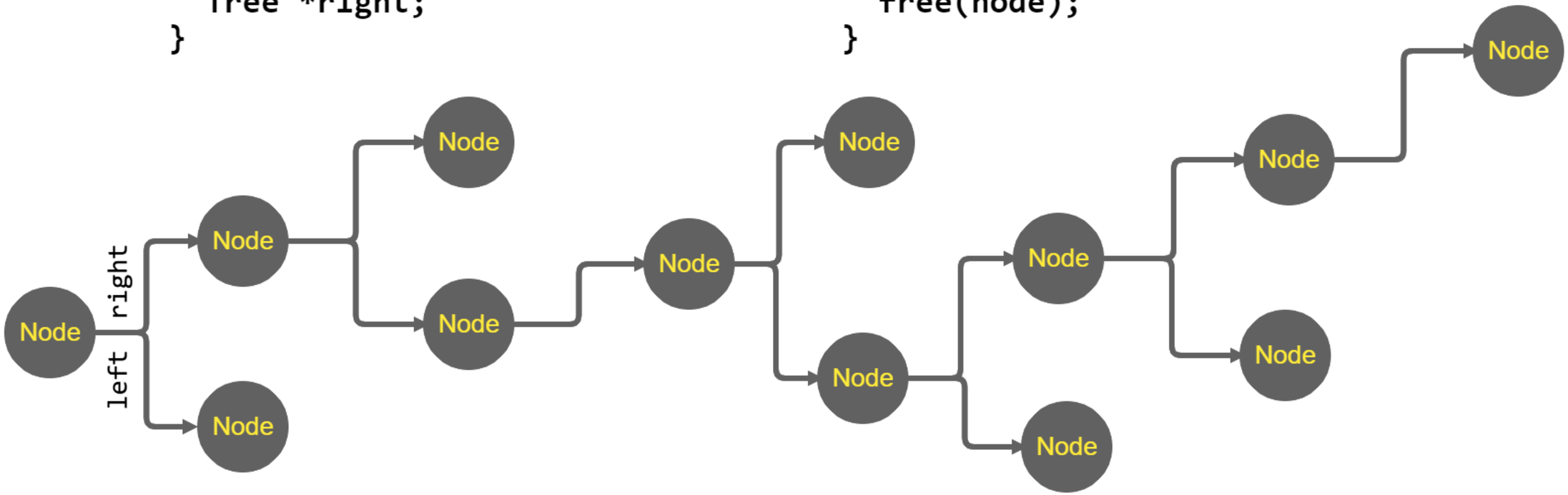
Stack spike

Before RA Grant

Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

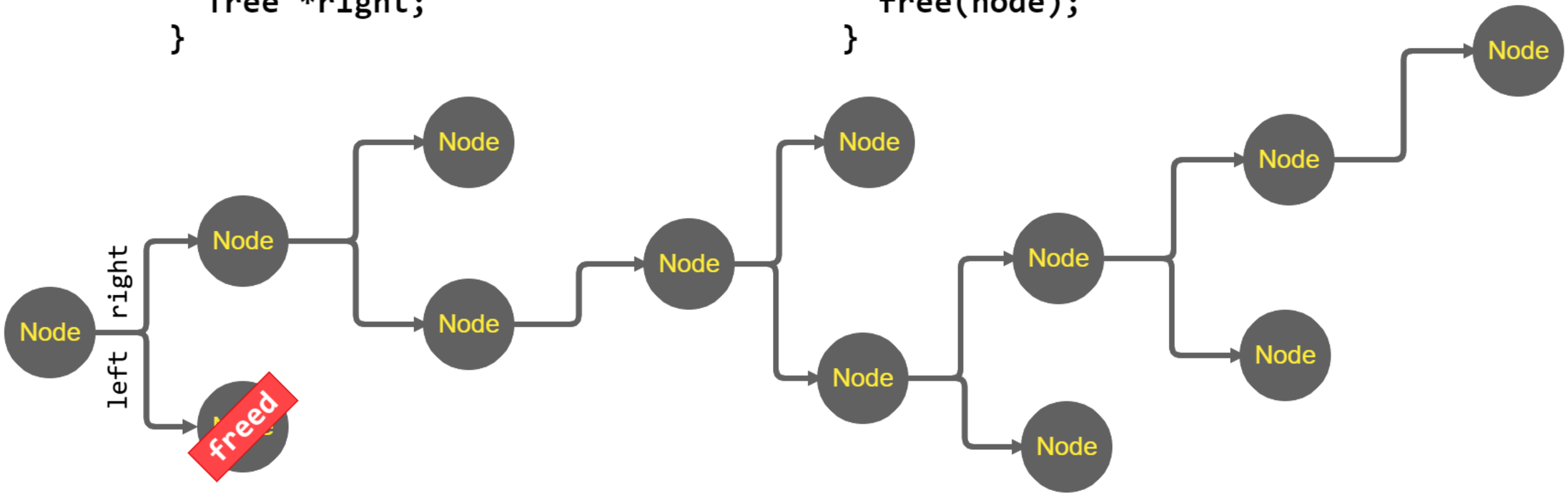
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

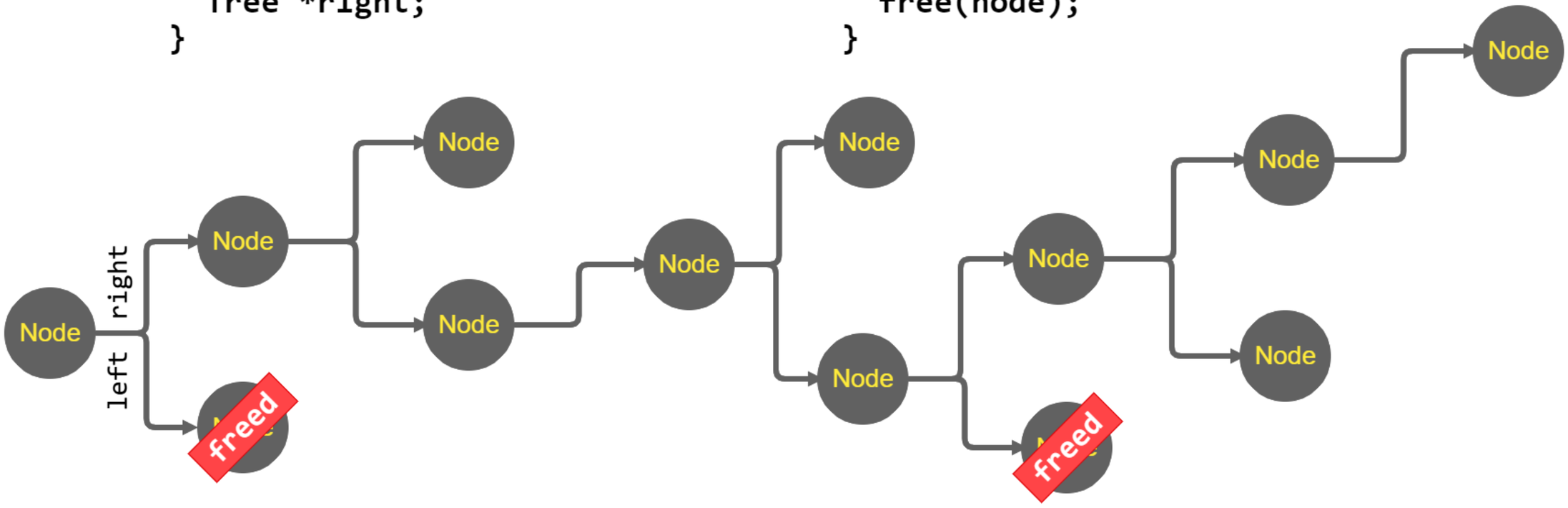
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

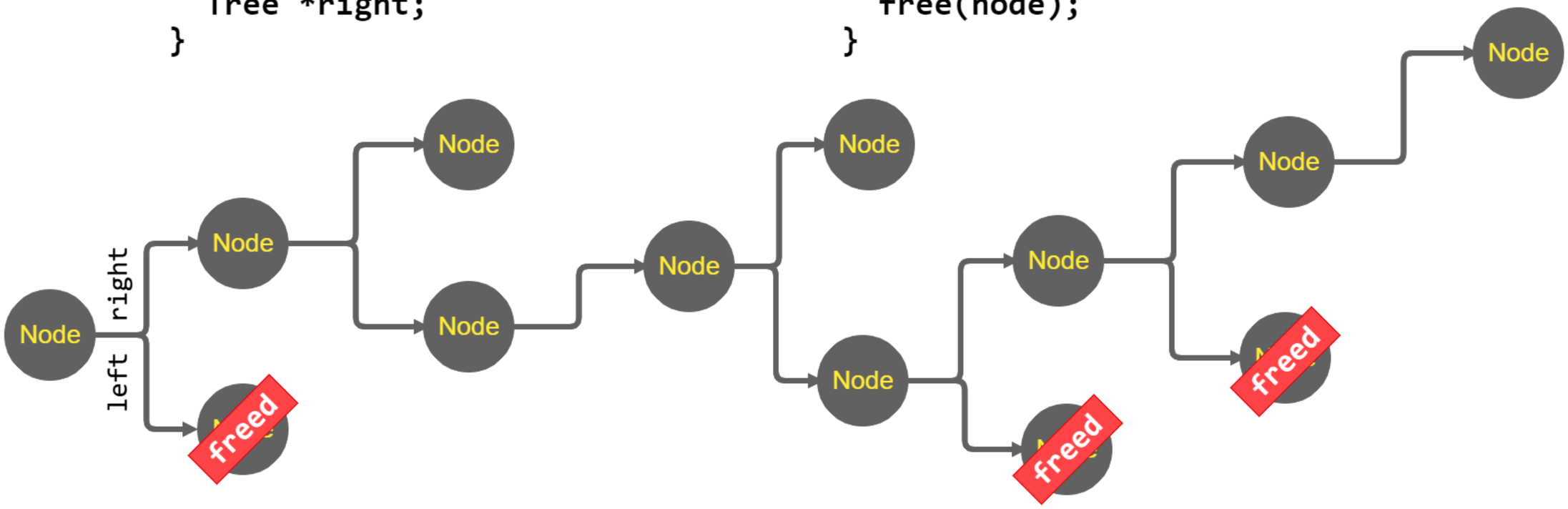
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



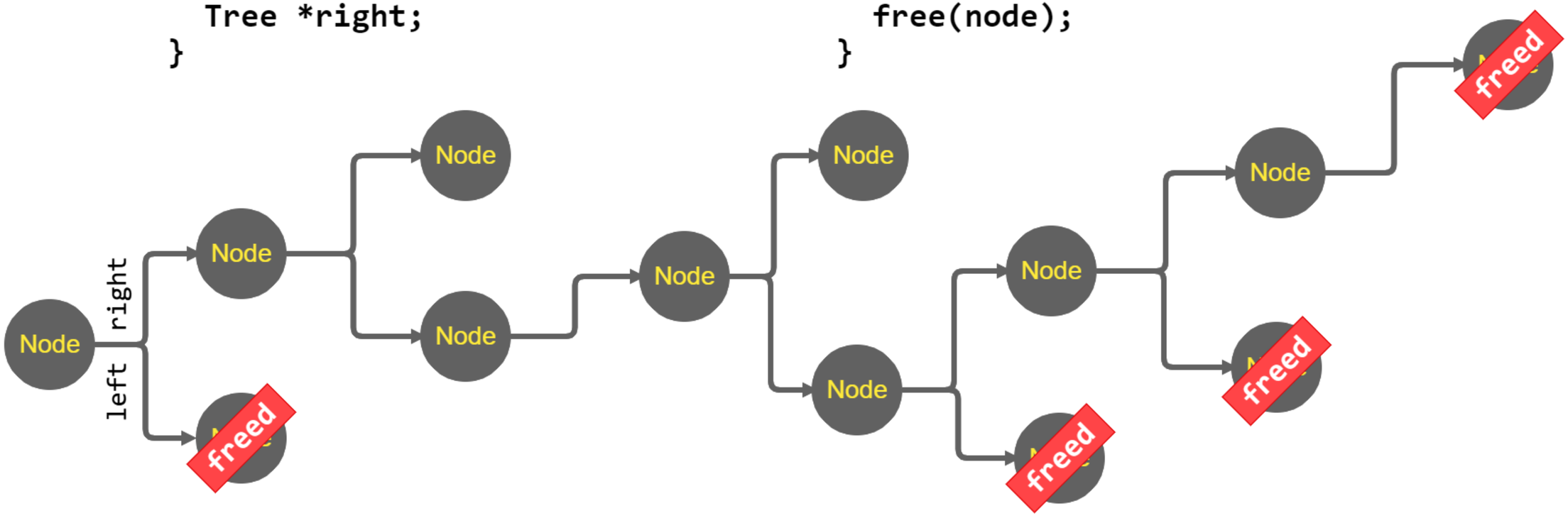
Stack spike

Before RA Grant

Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

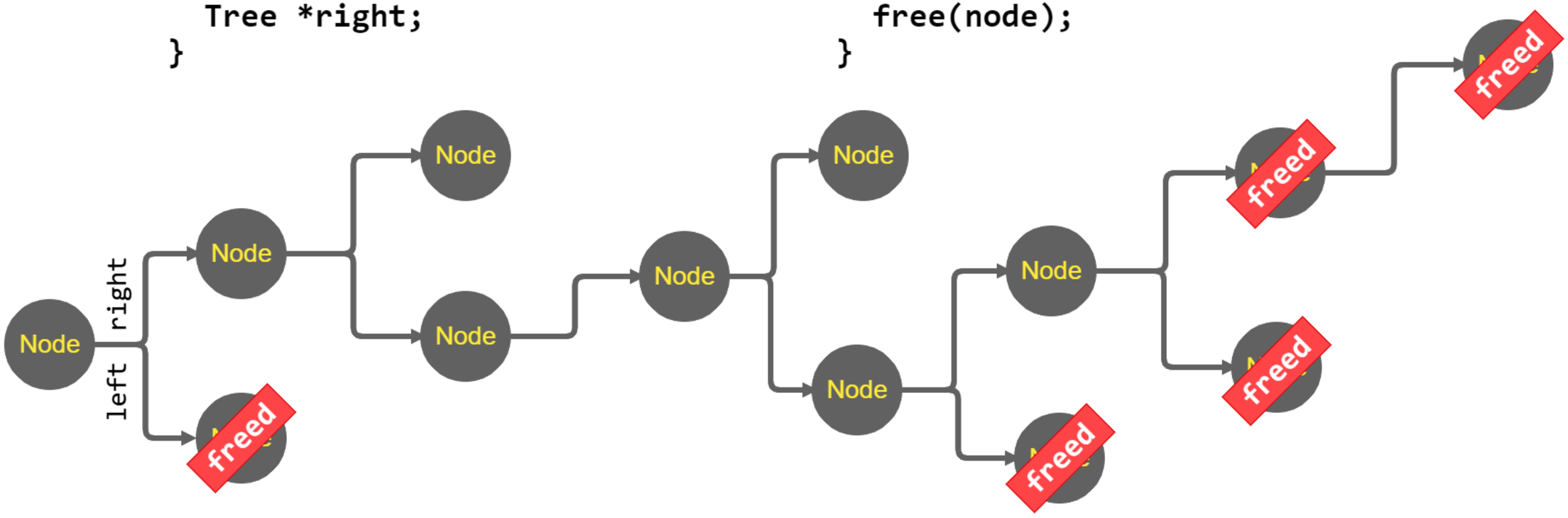
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

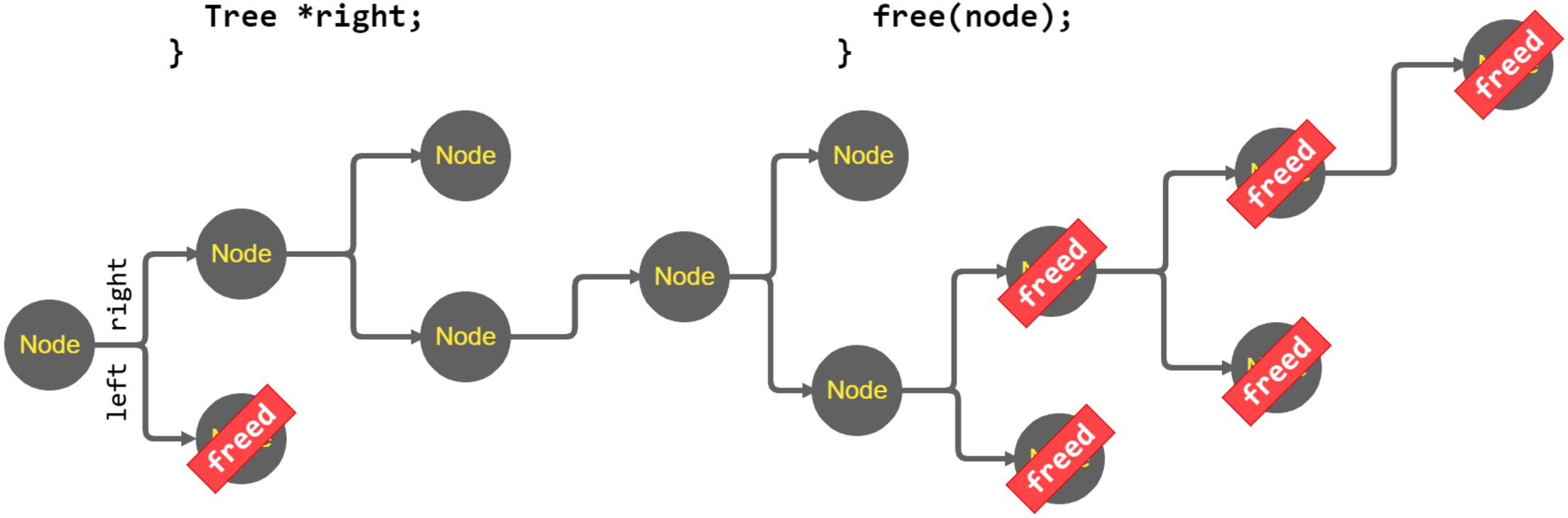
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

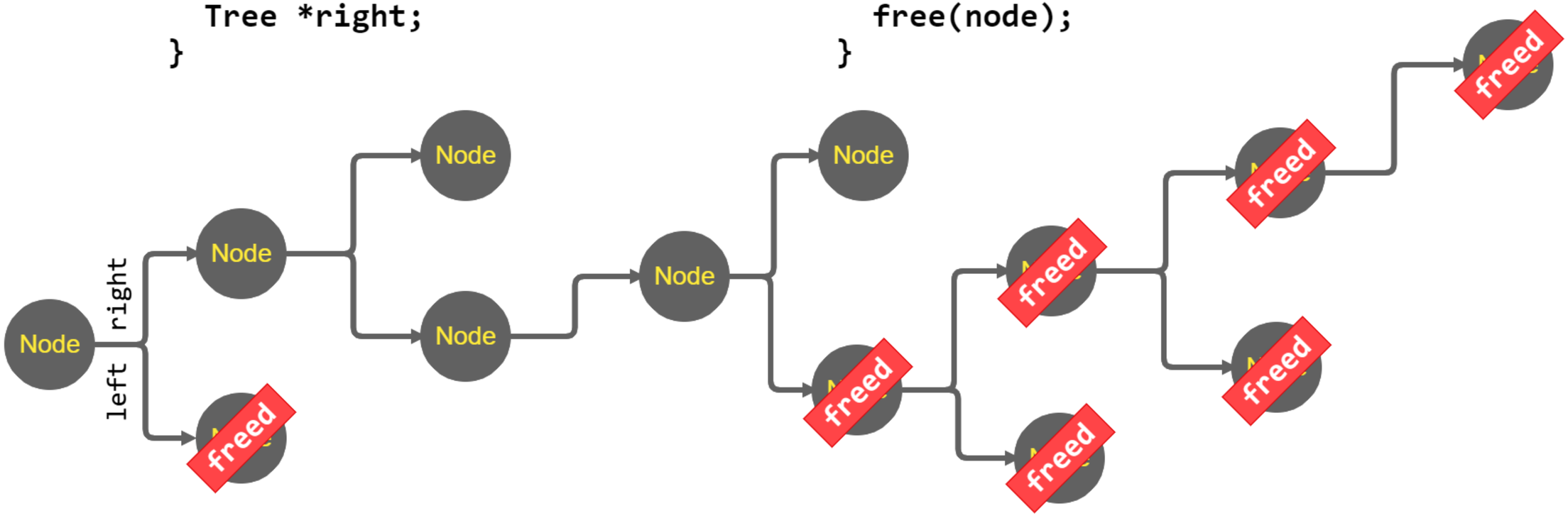
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

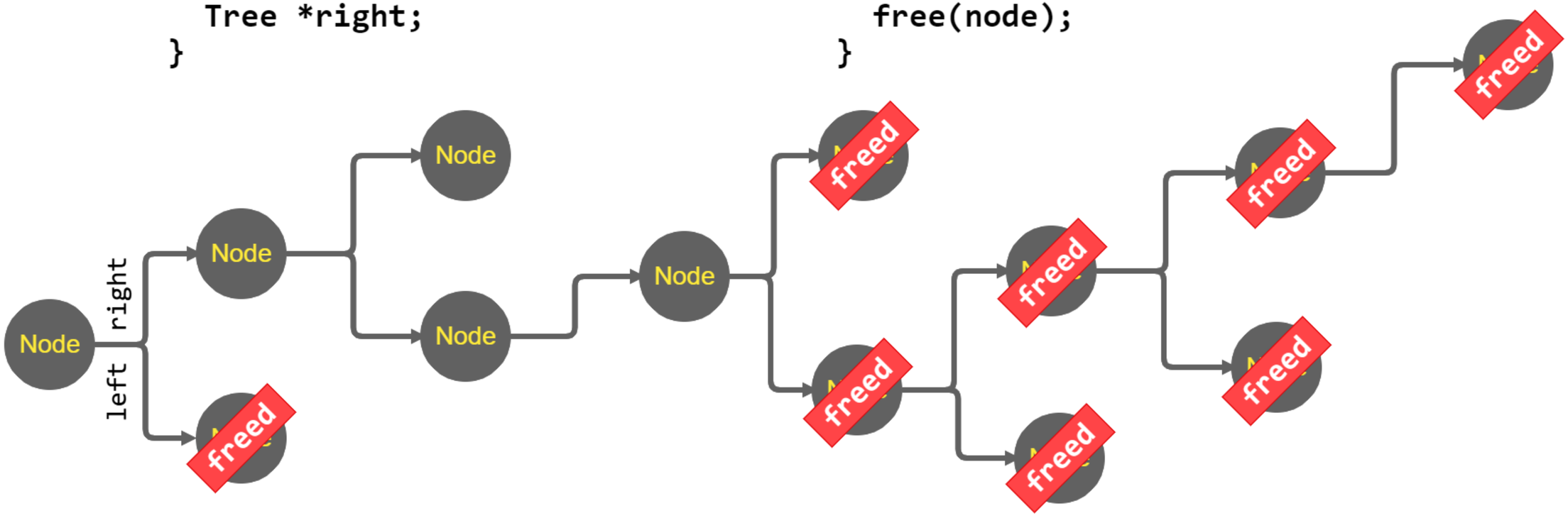
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

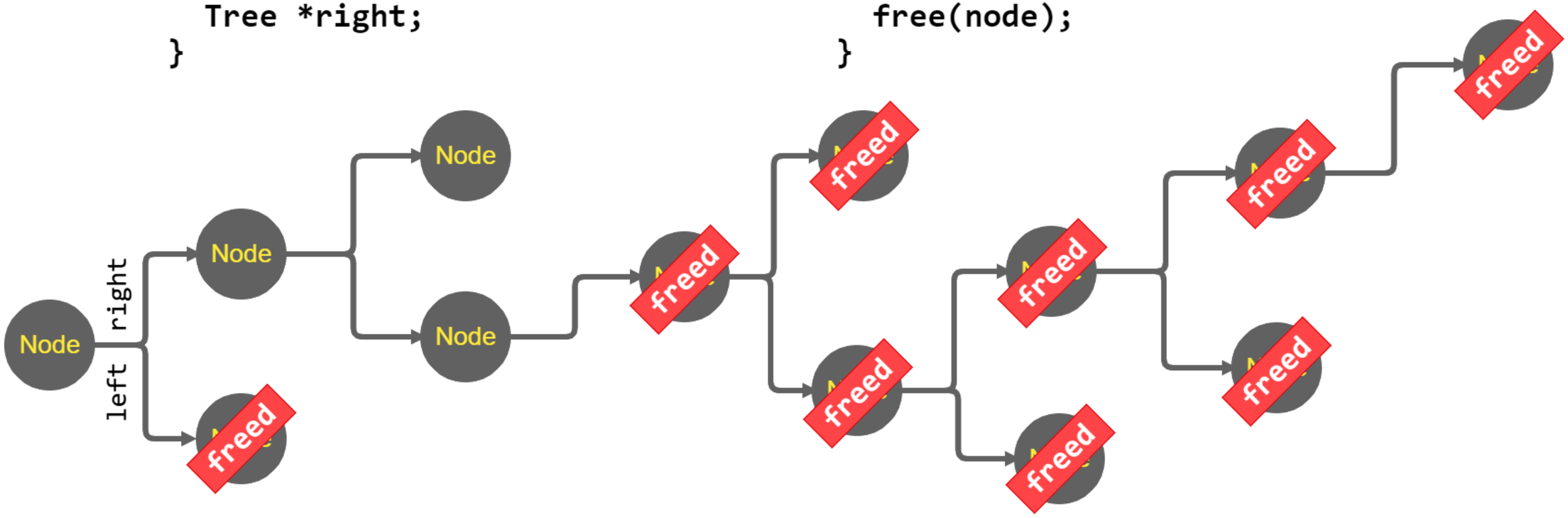
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

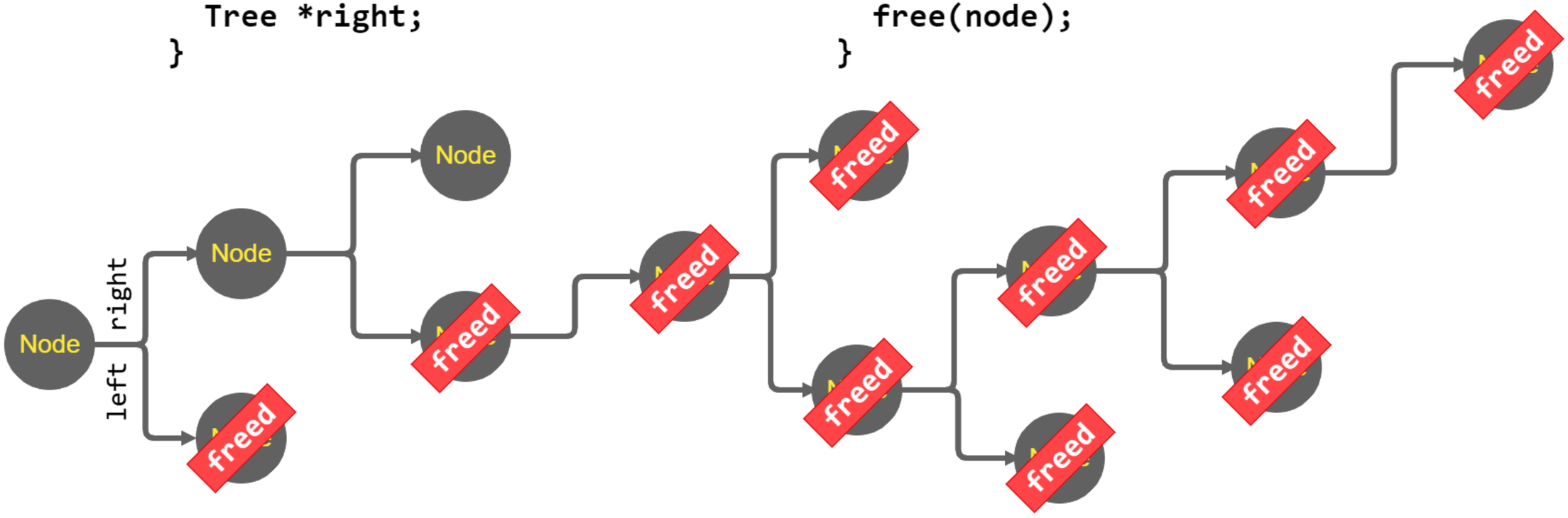
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

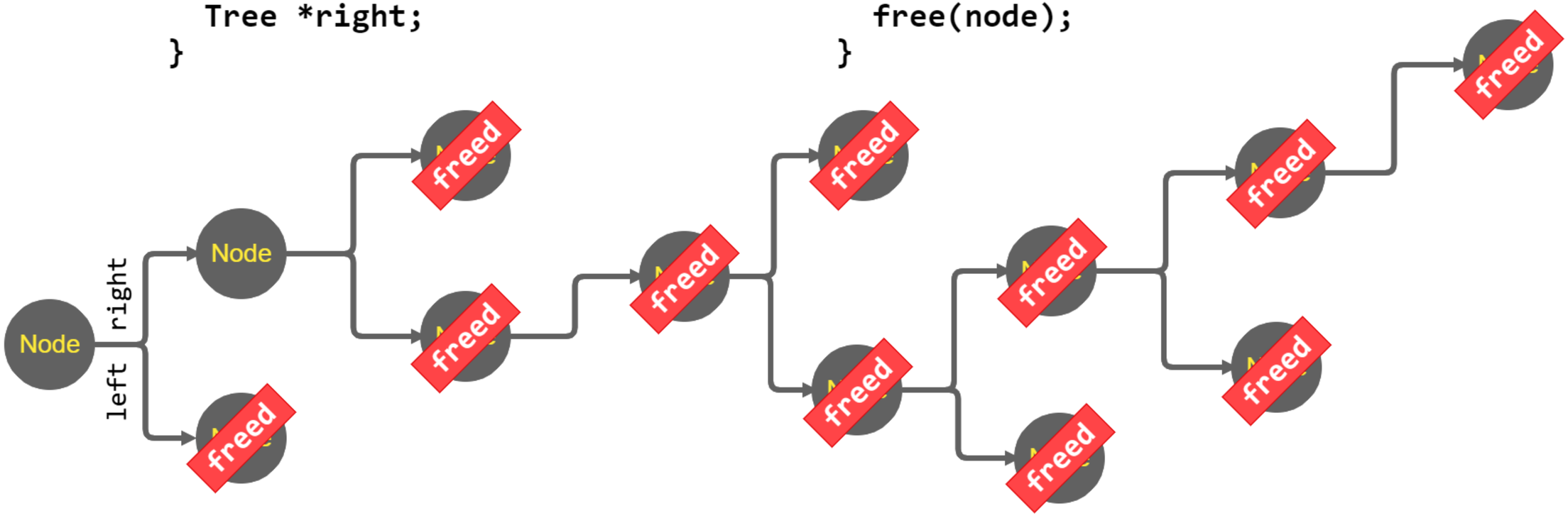
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

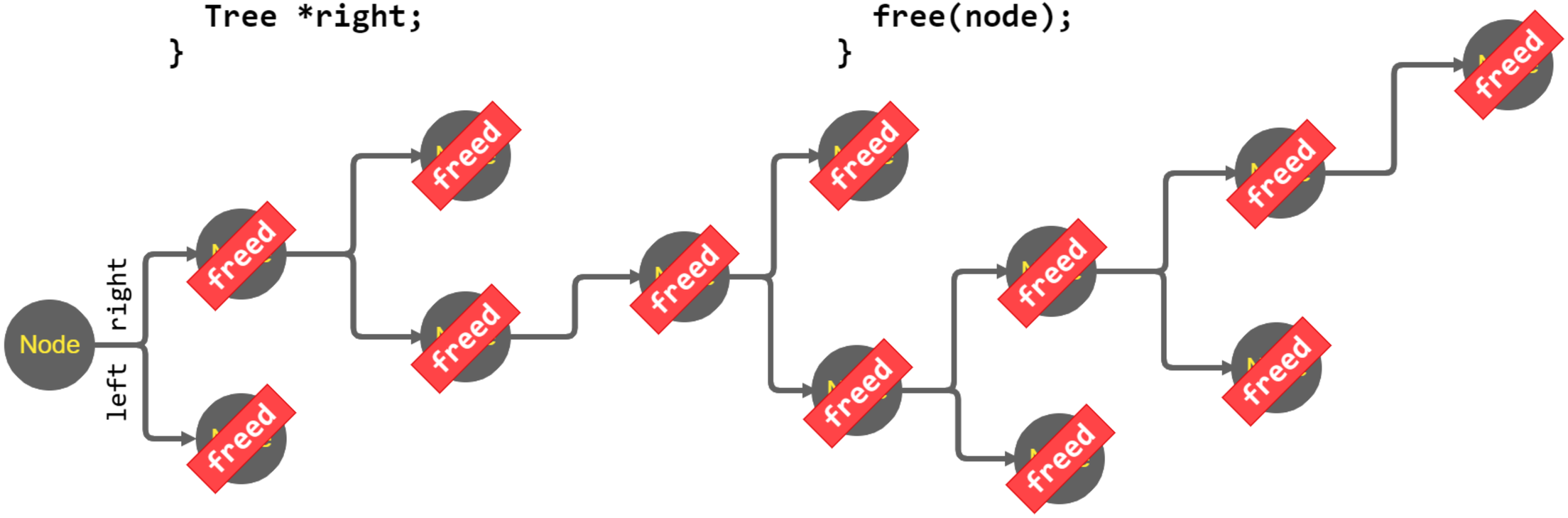
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

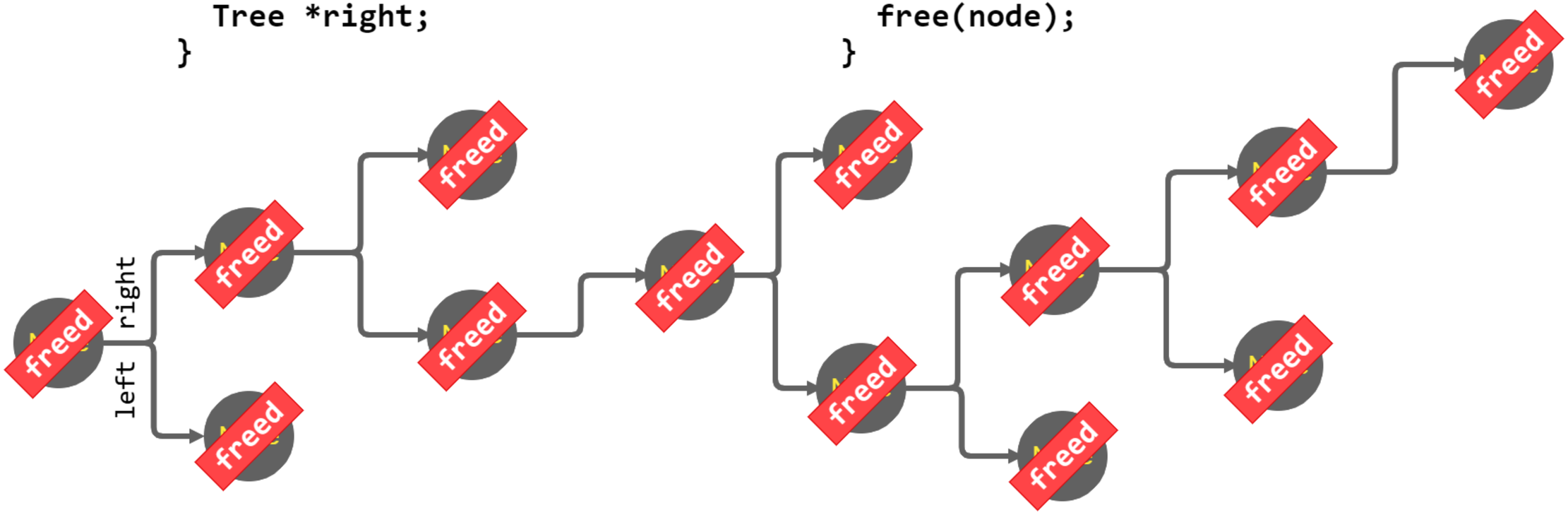
```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Stack spike

```
typedef struct _tree Tree;

struct _tree {
  Data *data;
  Tree *left;
  Tree *right;
}
```

```
void freeTree(Tree *node) {
  if (node == NULL) return;
  freeTree(node->left);
  freeTree(node->right);
  freeData(node->data);
  free(node);
}
```



Freeing in loop instead of recursion

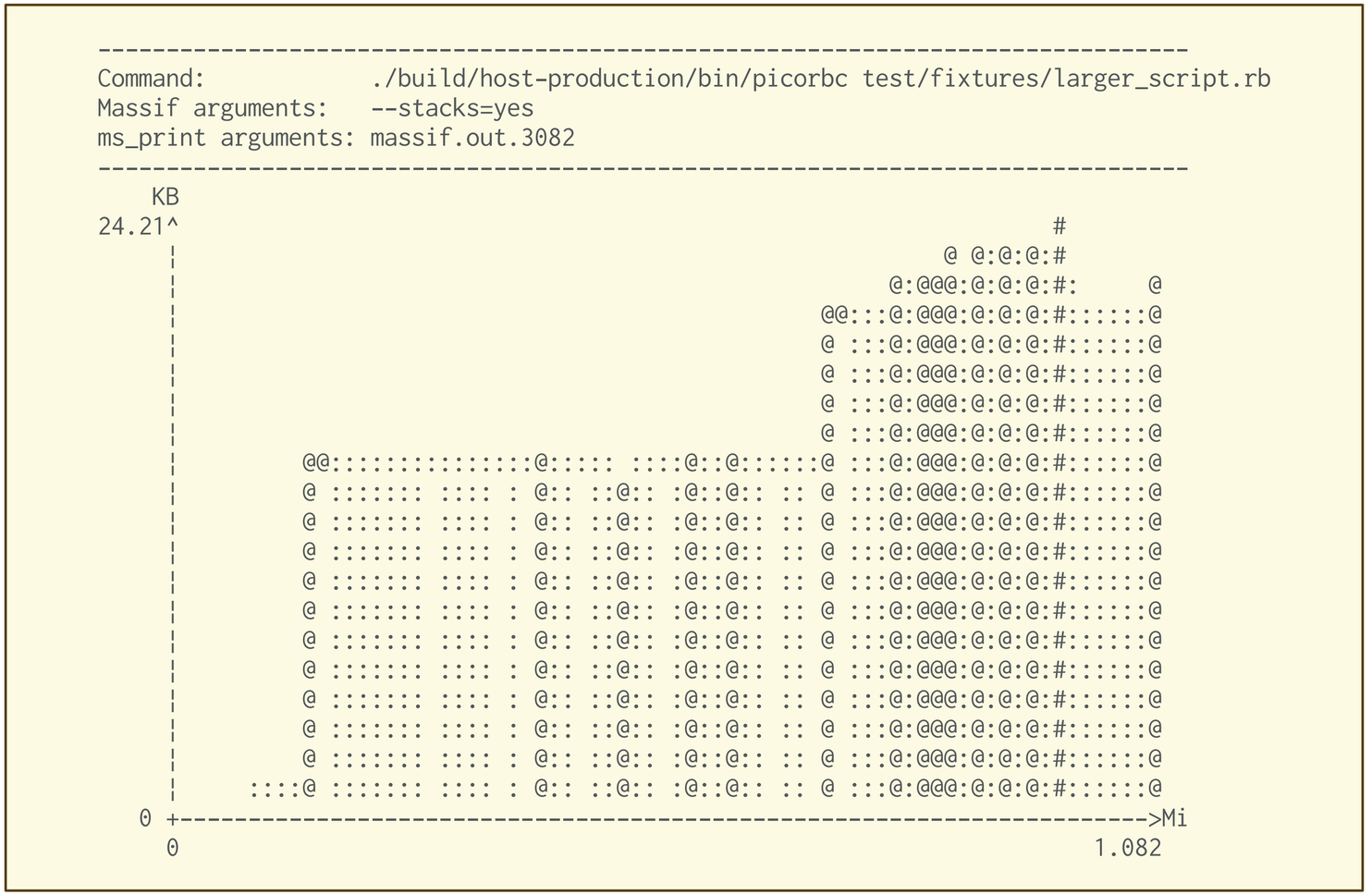
```c
// Recursion is elegant but a big eater
void freeList_in_recursion(List *list) {
  if (!list) return;
  freeList_in_recursion(list->next);
  free(list);
}

// Loop is somehow clumsy but thrifty
void freeList_in_loop(List *list) {
  List *next;
  while (list) {
    next = list->next;
    free(list);
    list = next;
  }
}
```

Freeing in loop instead of recursion

before Grant

Freeing in loop instead of recursion

after Grant



What happened?

- RAM usage when compiling a very small statement like `puts "Hello World!"` became bigger than before due to Pooled allocation
  - Because Pooled allocation allocates a fixed size of array of struct in advance
- However, when a Ruby script becomes larger, measures of reducing RAM consumption including Pooled allocation work effectively

PicoRuby Compiler

Future work

Killer Application

Programming languages need a killer application

C ... UNIX

PHP ... WordPress

CRuby ... Rails

How about PicoRuby?


F1 F2 F3 F4 F5 F6
Esc Q W E R T
A S D F G
PgDn Z X C V B
Alt
2@
F7 F8 F9 F10 F11 F12
Y U I O P
H J K L
N M
Fn
PgUp
sparkfun
Pro Micro
RP2040
SW35
SW70

To be continued on RubyKaigi Takeout 2021 or RubyConf 2021

🤞 if proposals.any?(&:accepted?) 🤞

Thank you!

Monstarlab ：：