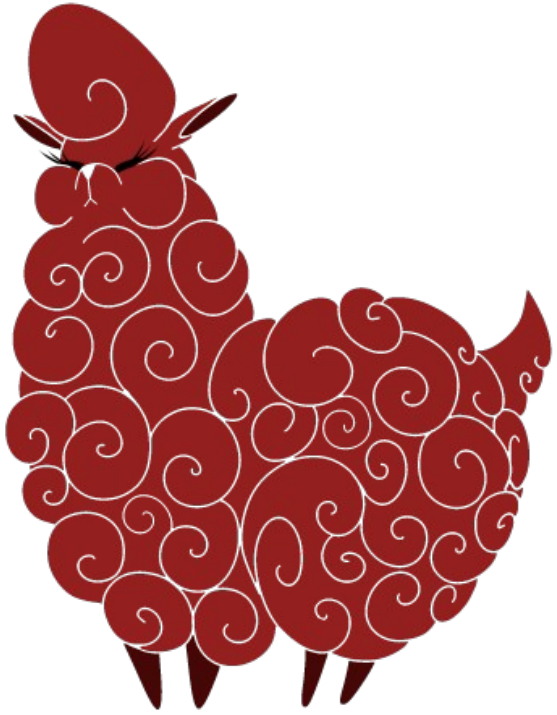
Lisp ”images” in LBM



Bo Joel Svensson

A Chalmers FP-Talk!

2025

LispBM (LBM)

- Lisp like language for microcontrollers.
  - (,), car, cdr, (+ 1 2)
- With some built-in functionality inspired by Erlang.
  - Pattern matching, message passing, process monitoring.

Target platforms

- Flexible, but mostly
  - STM32 (ARM) microcontrollers.
    - 192KB ram, up to 1MB flash.
  - ESP32C3 (RISC-V) microcontrollers.
    - 400KB ram, 4MB flash.
- Also runs fine on
  - X86, Raspberry PI ...

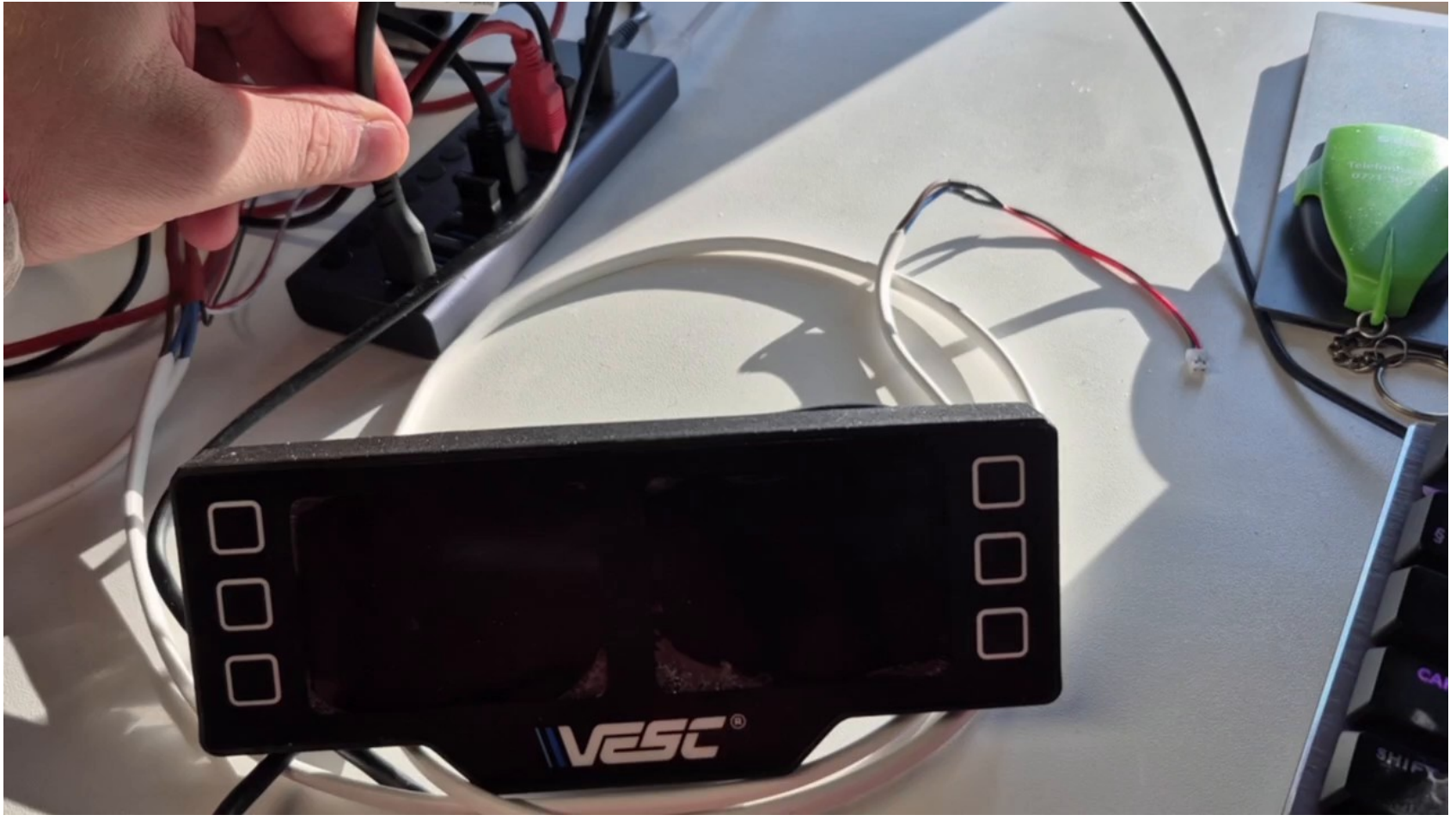
Purpose

- Integrate into existing embedded application to provide a scripting layer.
  - Sandboxed.
  - Not the only thing using FLASH/RAM.

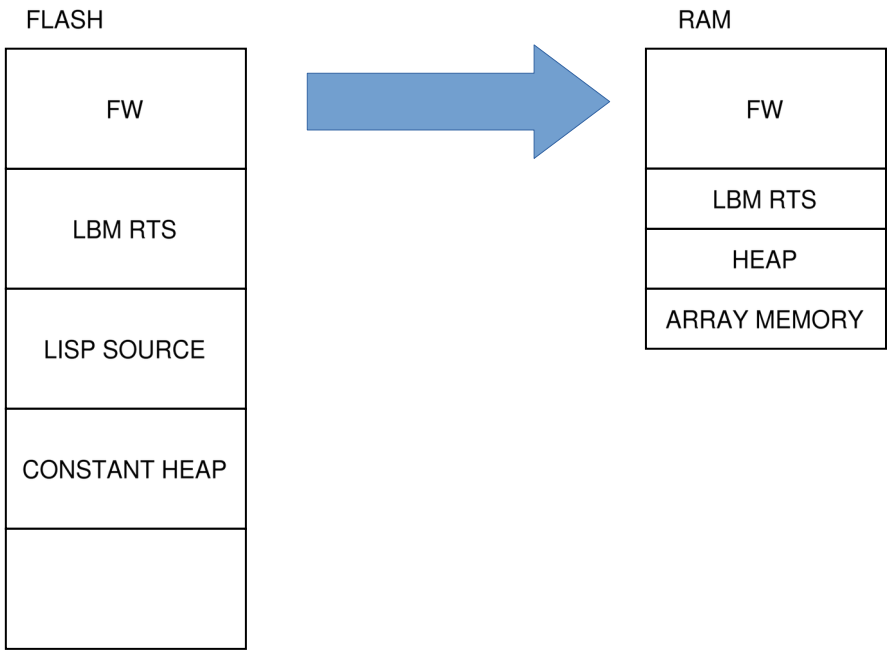
Problems!

- Our programs are larger than our RAM.
- Flash memory has limitations.
- Booting is slow.

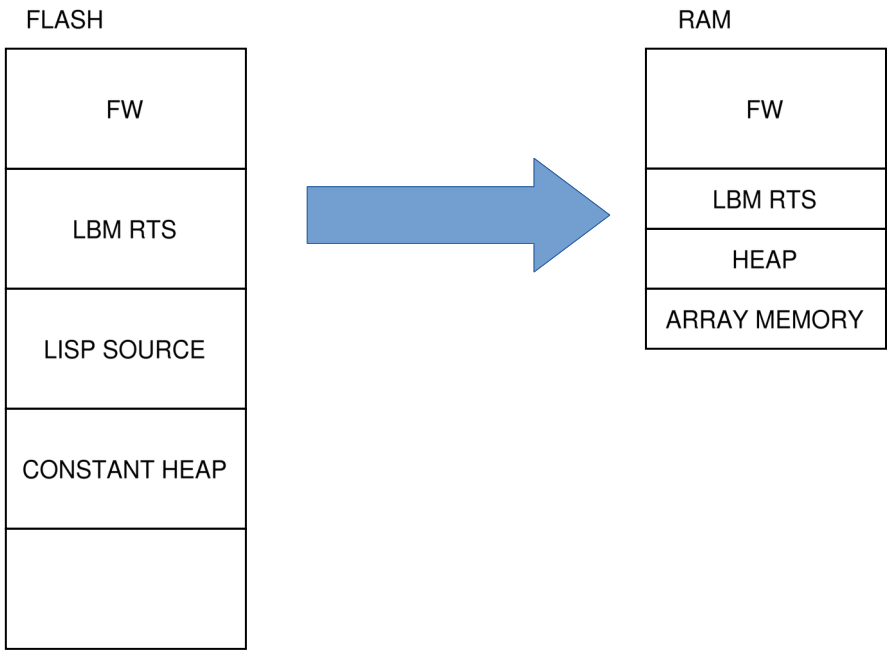
Booting an LBM application



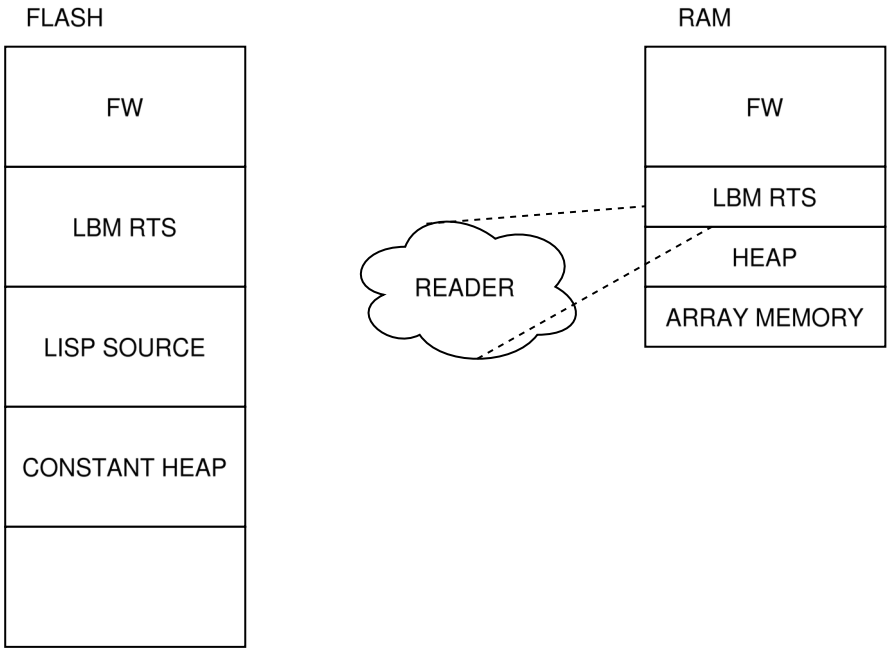
Boot process



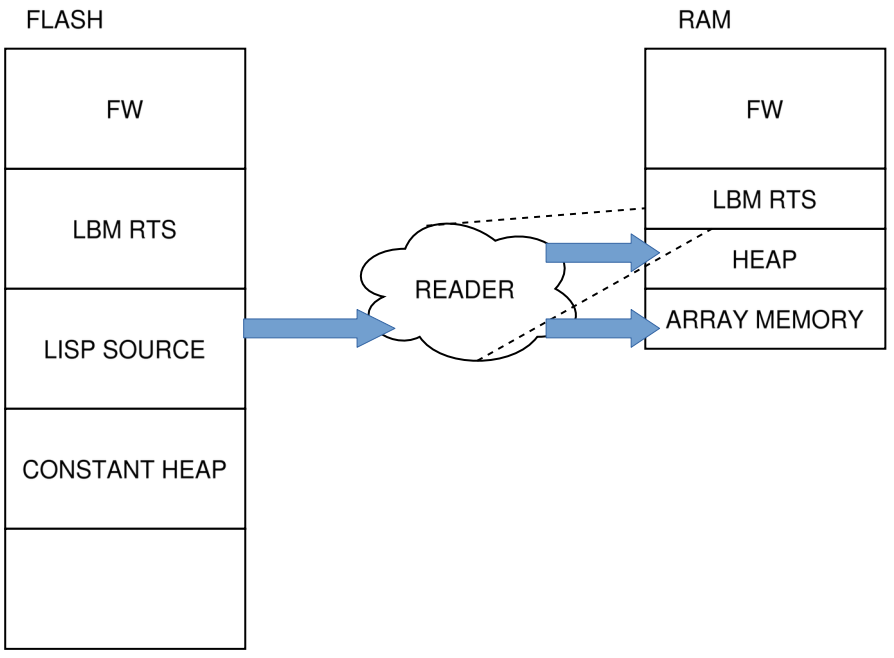
Boot process



Boot process



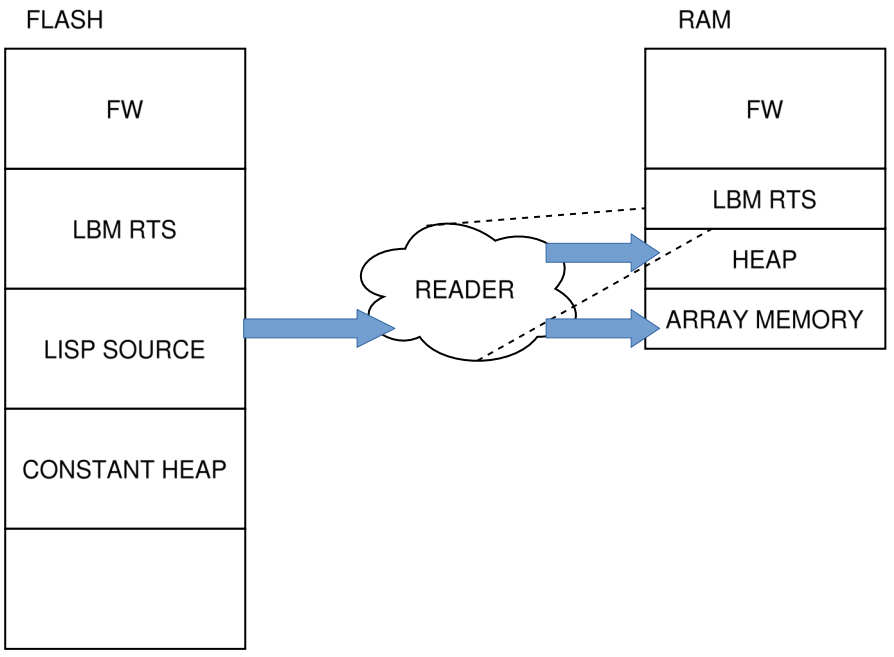
Boot process

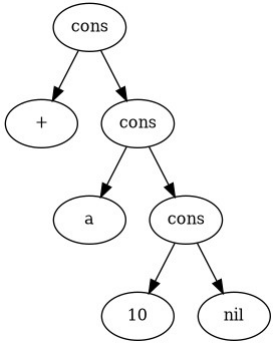


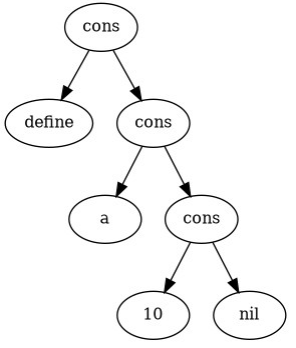
Boot process

(define a 10)

(+ a 10)



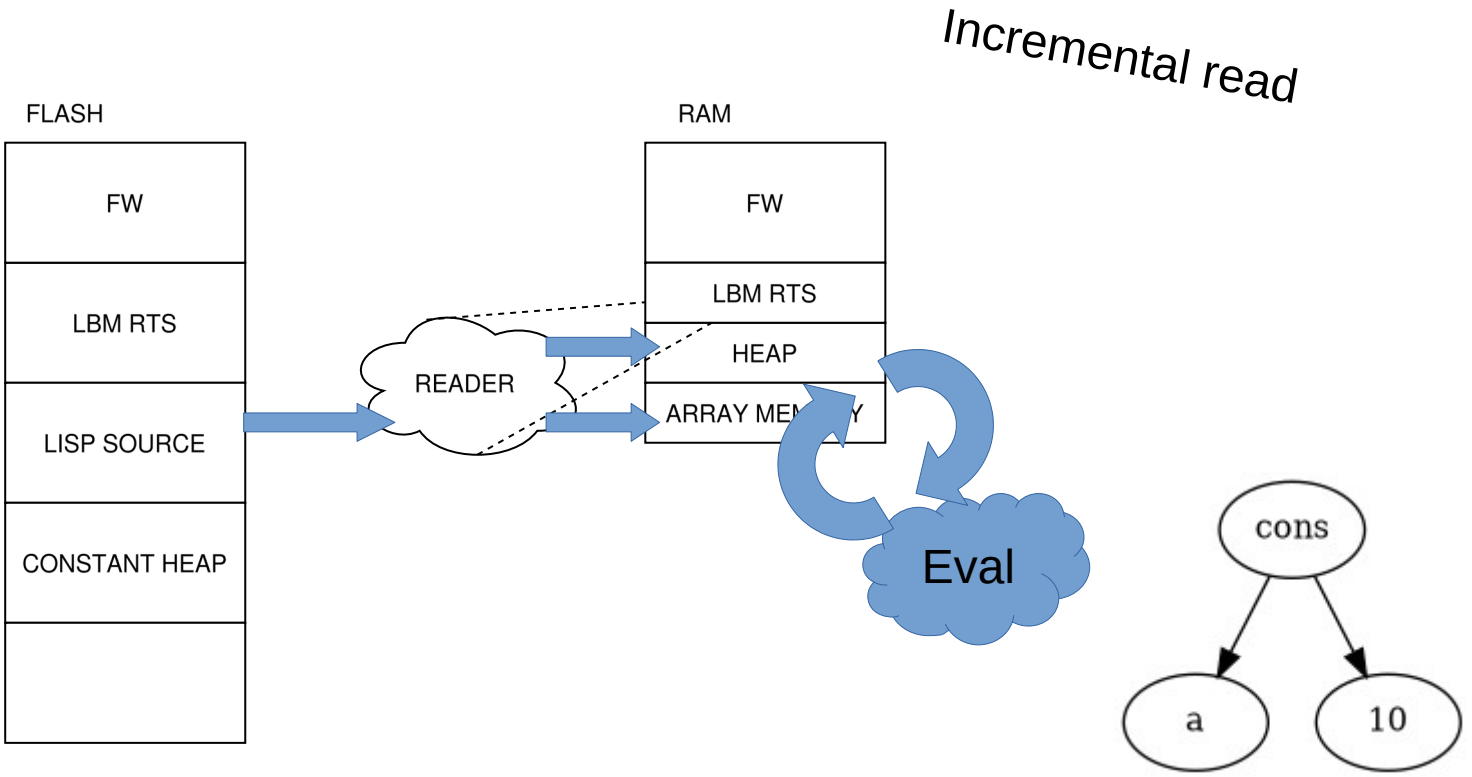




Boot process

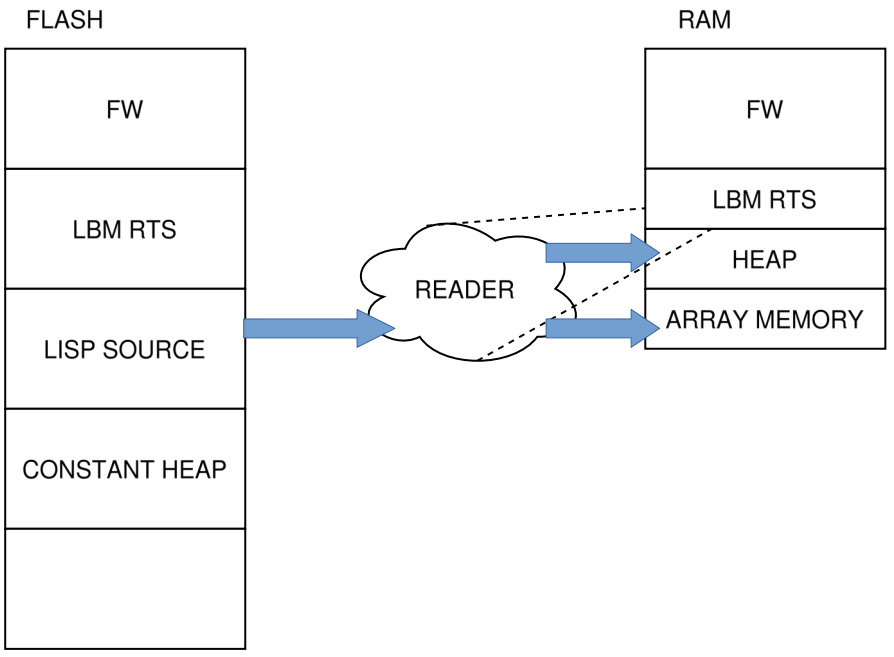
(define a 10)

(+ a 10)

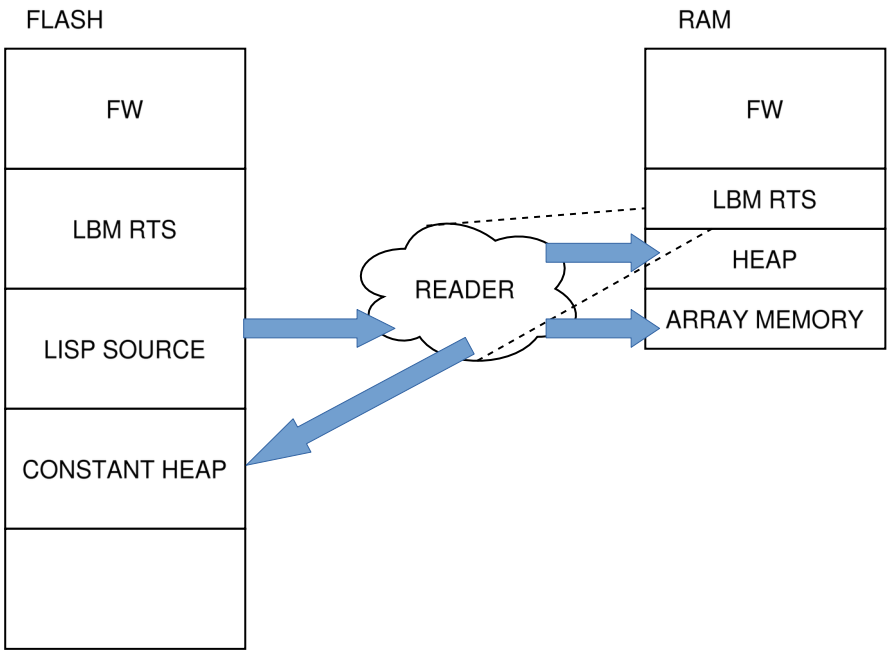


Boot process

May require more heap than we have!

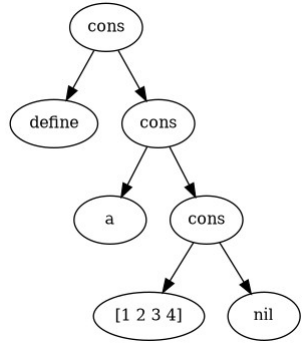


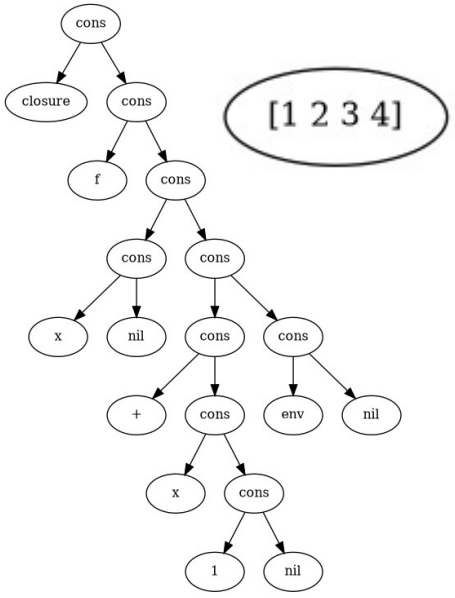
Boot process



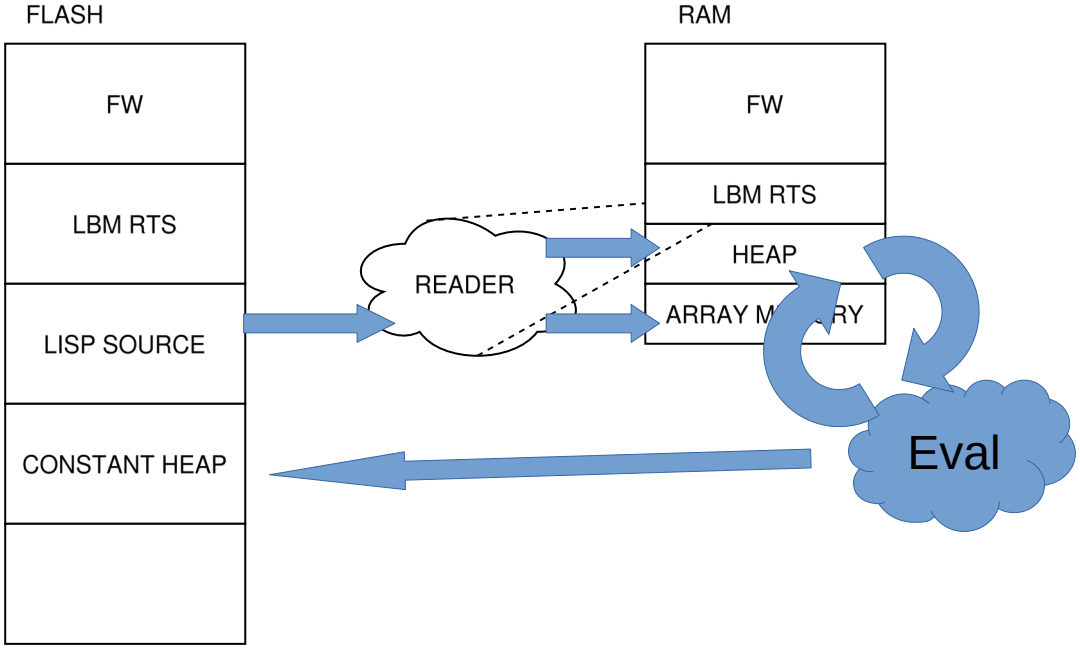
Boot process

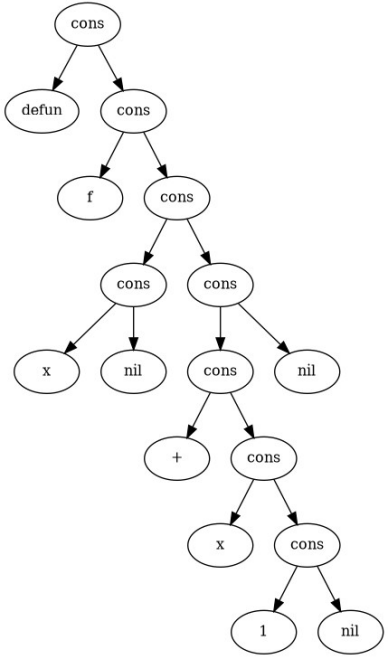
```
@const-start
(define a [1 2 3 4])
(defun f (x) (+ x 1))
@const-end
```





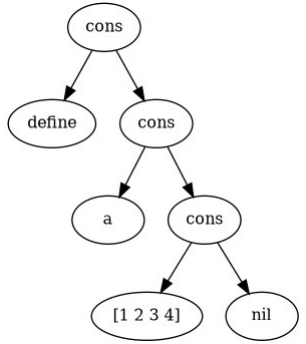
[1 2 3 4]

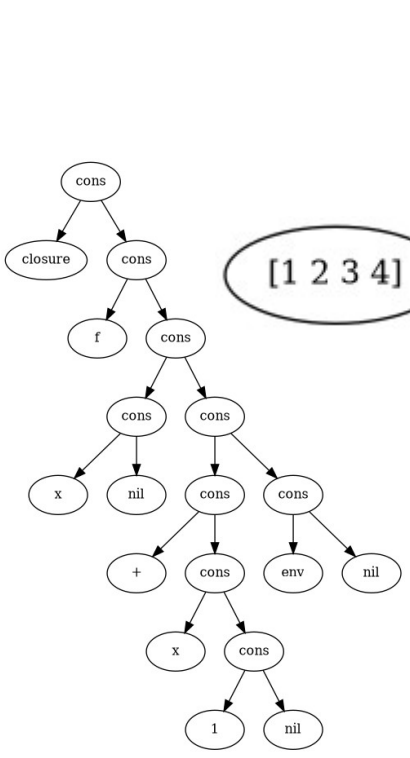


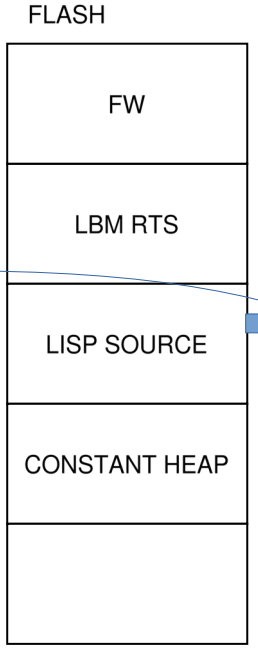


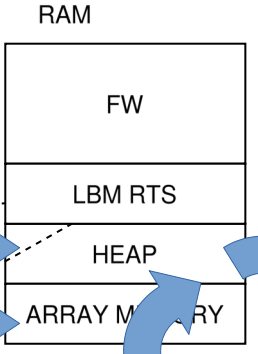
Boot process

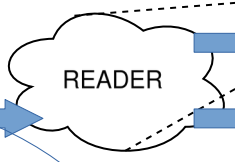
```
@const-start
(define a [1 2 3 4])
(defun f (x) (+ x 1))
@const-end
```

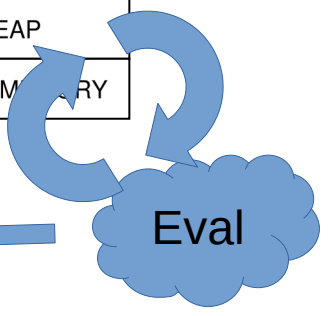


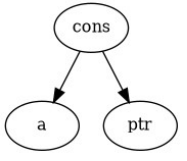


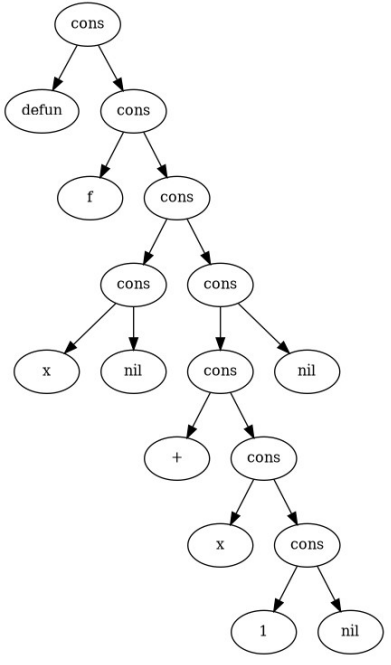






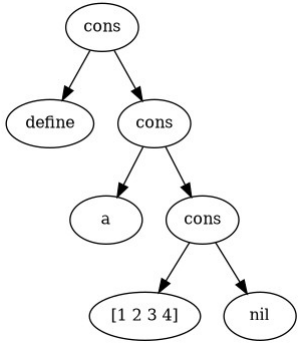


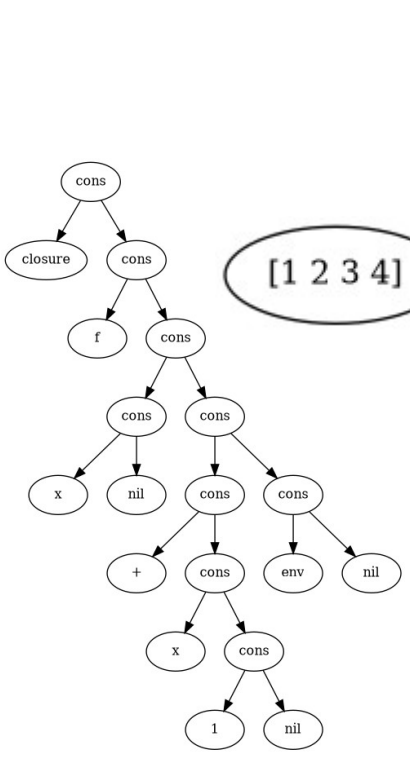




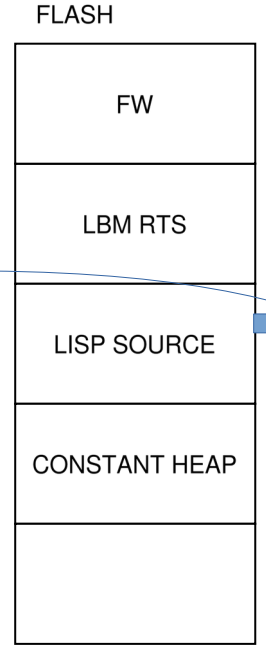
Boot process

```
@const-start
(define a [1 2 3 4])
(defun f (x) (+ x 1))
@const-end
```

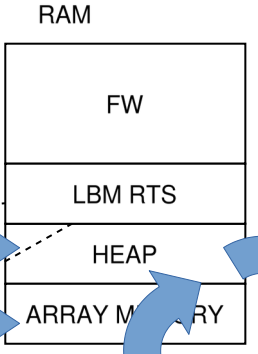




[1 2 3 4]

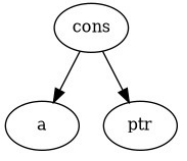


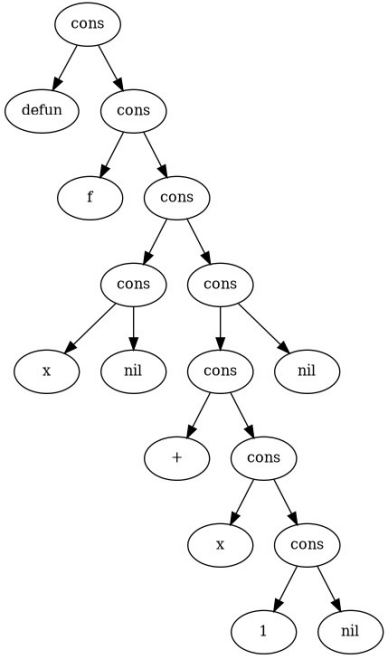
READER



Eval

Flash is a bit tricky!





Why are constant blocks tricky

```
(spawn f)

(if (system-initialized)
    ()
    (initialize))

(define randomly-long-list (range (random 10)))

@const-start
(stuff)
@const-end
```

Summary so far

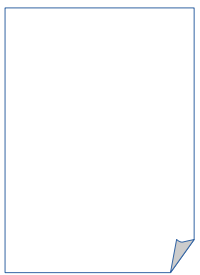
- The boot process is slow!
- Constant heap is a bit awkward.
  - Careful programming needed!

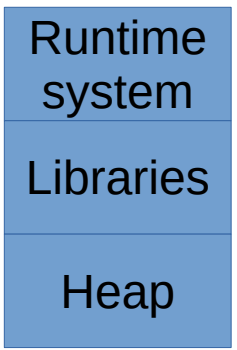
Images to the rescue!

- Image based development.
  - Smalltalk, SBCL.
  - Fun playful REPL-interaction based development.
  - (save-image)

The idea

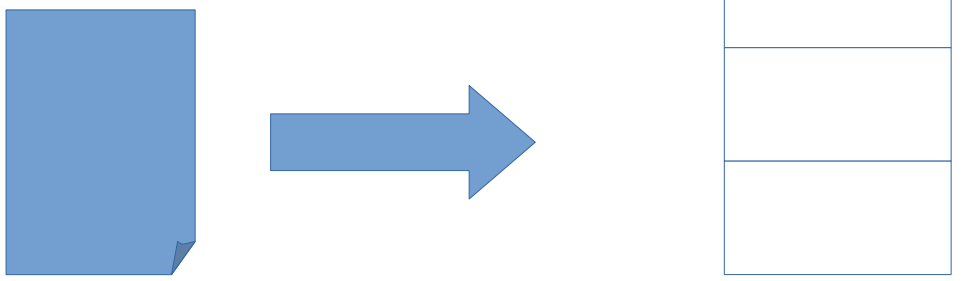
Distributable
binary blob





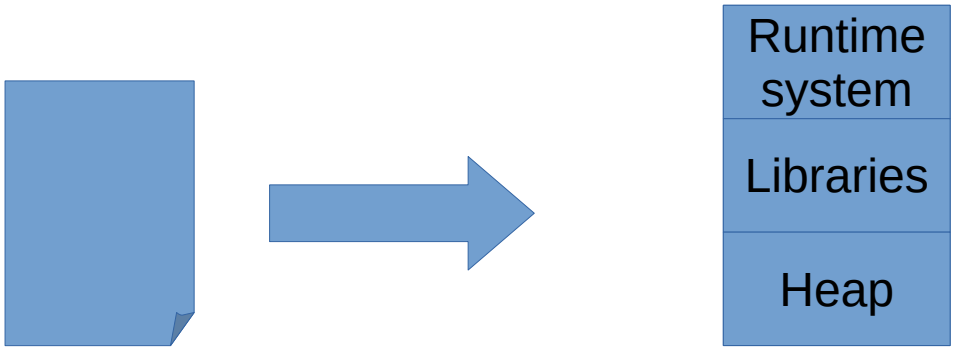
Restart

Distributable
binary blob



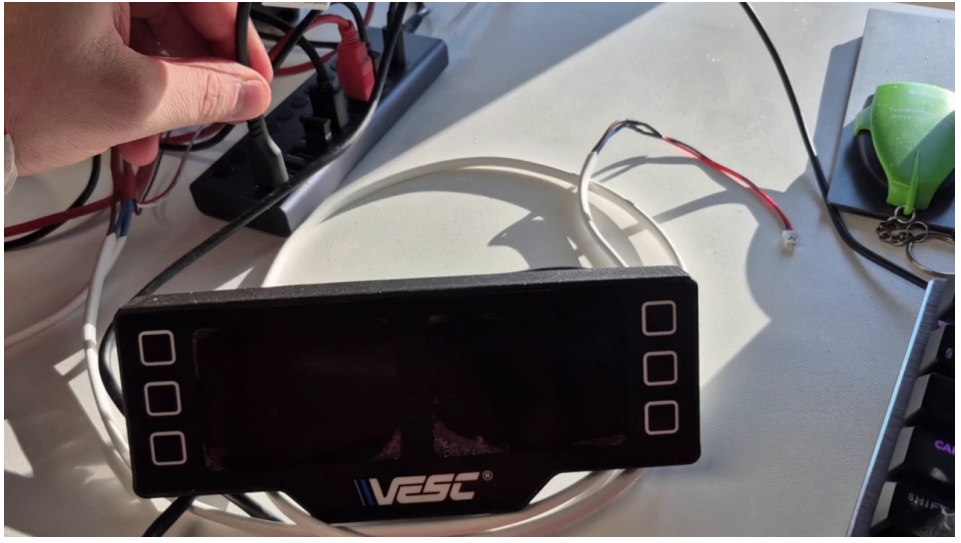
Restart

Distributable binary blob



Restarting a sytem in this way should be MUCH faster than that entire reader based bootup-process from earlier!


VESC


VESC

Problems!

- Not enough room for a complete copy of HEAP, arrays-memory and constant-heap.
- RTS data-structures must be restored upon image-boot.
  - Symbol numbering, extensions
- Initialization of peripherals.

Decisions

- Not going to save all state.
  - Not saving threads.
  - Not even going to try to automatically reinitialize any peripherals.
  - Not going to save entire RAM heap to image.
  - Not going to save entire arrays memory to image.
- A Startup entry is saved in the image.
  - Reinitialize peripherals.
  - Start up threads.
- Save the values stored in the global environment.

```
(defun f () ... )
(defun g () ... )


(defun main () {
        (init-uart 115200)

        (spawn f)
        (spawn g)

        })

(image-save)
```

Decisions

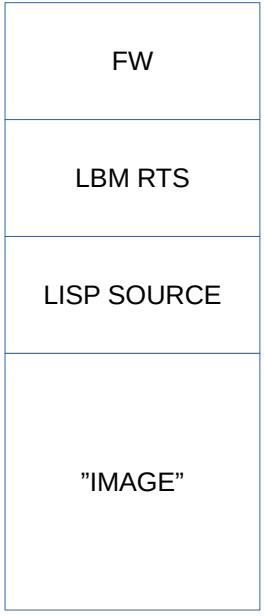
- Create the image incrementally
  - The constant heap is built-up inside of the image from the beginning.
  - RTS data-structures that needs to be restored upon image-boot are created inside the image storage by default.
    - Symbol string ↔ number mappings.
    - Extensions function pointer addresses.
    - Constant heap write position.

Image Structure

- The image is a collection of data fields written into flash.
- Duplicated data fields are allowed, where later fields have priority over earlier. Allows Incremental work towards same image.

```
#ifdef LBM64
#define IMAGE_INITIALIZED (uint32_t)0xBEEF4001
#else
#define IMAGE_INITIALIZED (uint32_t)0xBEEF2001
#endif

#define CONSTANT_HEAP_IX  (uint32_t)0x02
#define BINDING_CONST     (uint32_t)0x03
#define BINDING_FLAT      (uint32_t)0x04
#define SYMBOL_ENTRY      (uint32_t)0x06
#define SYMBOL_LINK_ENTRY (uint32_t)0x07
#define EXTENSION_TABLE   (uint32_t)0x08
#define VERSION_ENTRY     (uint32_t)0x09
```

The image



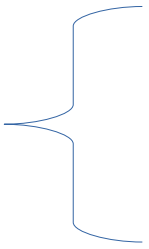
- Constant heap grows from bottom of image and upwards.
- All other data is added from the top and downwards in the image.
- When these two write pointers meet, image is full.

Creating an image

```
@const-start
(define a ...)
(define b ...)

(defun h ...)
@const-end

(define i ...)
(defun f () ... )
(defun g () ... )


(defun main () {
        ...
       })

(image-save)
```

Creating an image

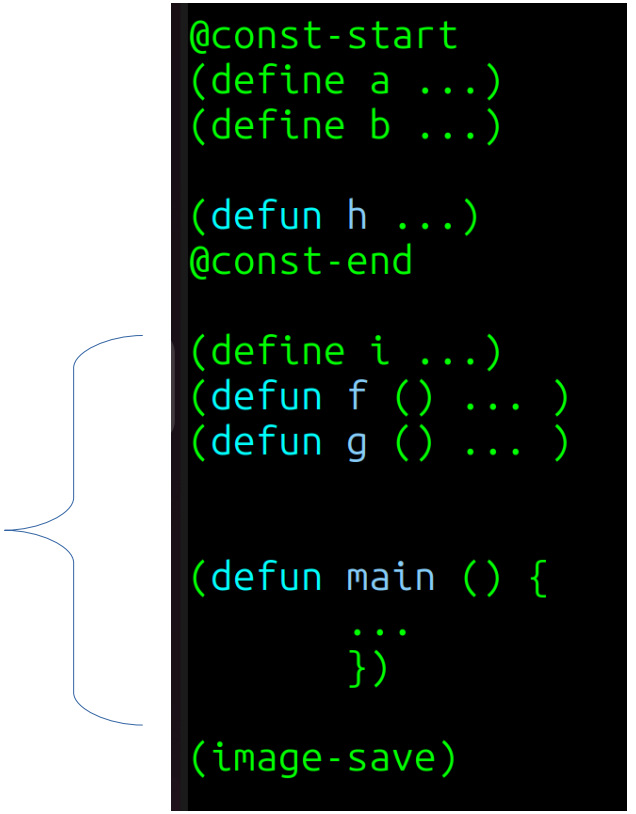
Source is read and evaluated, data-structures built in constant heap.



```
@const-start
(define a ...)
(define b ...)

(defun h ...)
@const-end

(define i ...)
(defun f () ... )
(defun g () ... )


(defun main () {
        ...
       })

(image-save)
```

Creating an image

Source is read and evaluated, data-structures built in RAM heap.



```
@const-start
(define a ...)
(define b ...)

(defun h ...)
@const-end

(define i ...)
(defun f () ... )
(defun g () ... )


(defun main () {
        ...
       })

(image-save)
```

Creating an image

```
@const-start
(define a ...)
(define b ...)

(defun h ...)
@const-end

(define i ...)
(defun f () ... )
(defun g () ... )


(defun main () {
        ...
       })

(image-save)
```

Saves RAM heap structures to image (and more)

Typical Image-save

```
lbm_value ext_image_save(lbm_value *args, lbm_uint argn) {
  (void) args;
  (void) argn;

  bool r = lbm_image_save_global_env();

  lbm_uint main_sym = ENC_SYM_NIL;
  if (lbm_get_symbol_by_name("main", &main_sym)) {
    lbm_value binding;
    if ( lbm_global_env_lookup(&binding, lbm_enc_sym(main_sym))) {
      if (lbm_is_cons(binding) && lbm_car(binding) == ENC_SYM_CLOSURE) {
        goto image_has_main;
      }
    }
  }
  lbm_set_error_reason("No main function in image\n");
  return ENC_SYM_EERROR;
 image_has_main:
  r = r && lbm_image_save_extensions();
  r = r && lbm_image_save_constant_heap_ix();
  return r ? ENC_SYM_TRUE : ENC_SYM_NIL;
}
```

```
bool lbm_image_save_global_env(void) {
  lbm_value *env = lbm_get_global_env();
  if (env) {
    for (int i = 0; i < GLOBAL_ENV_ROOTS; i ++) {
      lbm_value curr = env[i];
      while(lbm_is_cons(curr)) {
        lbm_value name_field = lbm_caar(curr);
        lbm_value val_field  = lbm_cdr(lbm_car(curr));

        if (lbm_is_constant(val_field)) {
          write_u32(BINDING_CONST, &write_index, DOWNWARDS);
          write_lbm_value(name_field, &write_index, DOWNWARDS);
          write_lbm_value(val_field, &write_index, DOWNWARDS);
        } else {
          int fv_size = image_flatten_size(val_field);
          if (fv_size > 0) {
            fv_size = (fv_size % 4 == 0) ? (fv_size / 4) : (fv_size / 4) + 1; // num 32bit words
            if ((write_index - fv_size) <= (int32_t)image_const_heap.next) {
              return false;
            }
            write_u32(BINDING_FLAT, &write_index, DOWNWARDS);
            write_u32((uint32_t)fv_size , &write_index, DOWNWARDS);
            write_lbm_value(name_field, &write_index, DOWNWARDS);
            write_index = write_index - fv_size;  // subtract fv_size
            if (image_flatten_value(val_field)) { // adds fv_size backq
              // TODO: What error handling makes sense?
              fv_write_flush();
            }
            write_index = write_index - fv_size - 1; // subtract fv_size
          } else {
            return false;
          }
        }
        curr = lbm_cdr(curr);
      }
    }
    return true;
  }
  return false;
}
```

Boot

- Check if there is an image: 0xBEEF2001
  - Constant heap exists. Initialize
  - Start reading fields from top of image.

Restore environment

```
case BINDING_CONST: {
  lbm_uint bind_key = read_u32(pos);
  lbm_uint bind_val = read_u32(pos-1);
  pos -= 2;
  lbm_uint ix_key  = lbm_dec_sym(bind_key) & GLOBAL_ENV_MASK;
  lbm_value *global_env = lbm_get_global_env();
  lbm_uint orig_env = global_env[ix_key];
  lbm_value new_env = lbm_env_set(orig_env,bind_key,bind_val);

  if (lbm_is_symbol(new_env)) {
    return false;
  }
  global_env[ix_key] = new_env;
} break;
```

Restore environment

```
case BINDING_FLAT: {
  int32_t s = (int32_t)read_u32(pos);
  // size in 32 or 64 bit words.
  lbm_uint bind_key = read_u32(pos-1);
  pos -= 2;
  pos -= s;
  lbm_flat_value_t fv;
  fv.buf = (uint8_t*)(image_address + pos);
  fv.buf_size = (uint32_t)s * sizeof(lbm_uint); // GEQ to actual buf
  fv.buf_pos = 0;
  lbm_value unflattened;
  lbm_unflatten_value(&fv, &unflattened);
  if (lbm_is_symbol_merror(unflattened)) {
    lbm_perform_gc();
    lbm_unflatten_value(&fv, &unflattened);
  }

  lbm_uint ix_key  = lbm_dec_sym(bind_key) & GLOBAL_ENV_MASK;
  lbm_value *global_env = lbm_get_global_env();
  lbm_uint orig_env = global_env[ix_key];
  lbm_value new_env = lbm_env_set(orig_env,bind_key,unflattened);

  if (lbm_is_symbol(new_env)) {
    return false;
  }
  global_env[ix_key] = new_env;
  pos --;
} break;
```

Flatten and un-flatten

- Recursive serialisation and de-serialisation functions.
- We ran out of stack on the thread that performed serialisation and de-serialisation.

Flatten and un-flatten

- Recursive serialisation and de-serialisation functions.
- We ran out of stack on the thread that performed serialisation and de-serialisation.
- Replaced with pointer reversal algorithms!

Thoughts

- Boot is fast!
- Incremental work against image ”possible” but not ideal (duplication).
- Program source only read once, constant block determinism is not a problem.
- A “main” or “startup” function is needed.

Thoughts

- Flatten and un-flatten works only on non-cyclic lisp values.
- Flatten/un-flatten duplicates shared nodes.
- Don’t really need to store the lisp source in flash at all. Future work.

Duplication

```
(define a (list 1 2 3))
(define b (append (list 'a 'b) a))
(define c (append (list 'c 'd) a))
```

Conclusion

- Trade-off image size vs performance.
- Trade-off performance vs “correctness”.
- Work around limitations of target platforms.
  - RAM, FLASH, Constant code size concerns.
- Fun.


P
E
S
R