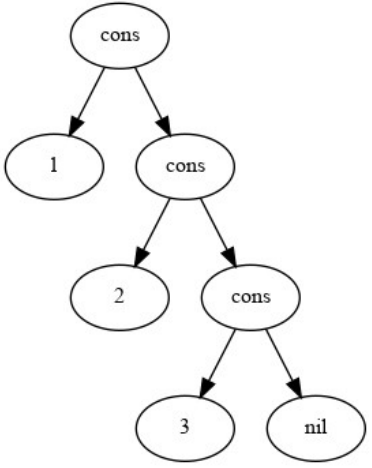
A linked list of cases in language design

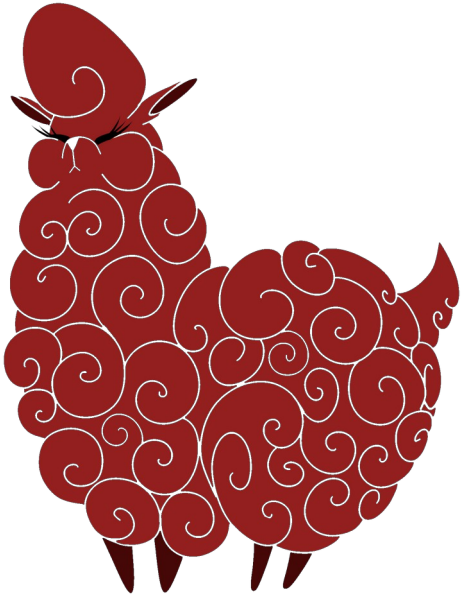
λ IFL 2026 - Gothenburg

Bo Joel Svensson
Benjamin Vedder



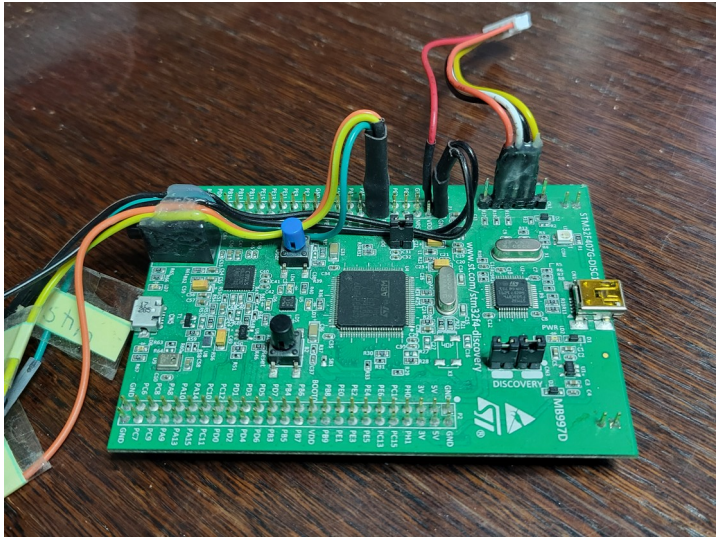
Background

- LispBM
  - A Lisp dialect
  - Concurrency
  - Message passing



Background

- For microcontrollers (MCUs)
  - 64 kB RAM and up.
  - Often a lot more flash memory (for example 1 MB).



Background

- VESC integration in December 2022
  - Scripting language for VESC motorcontrollers and peripherals.
  - Sandboxed.



100V 150A

Background





https://electricfox.de/en/Home/

Dec 2025: 17982 lines of Lisp



https://lind.surf/

A linked list of cases in language design

- Concurrency model switch
- Garbage collection troubles
- Compacting allocator
- Programs larger than RAM
- Fast boot and Lisp images



A Linked List of Cases in Language Design

Bo Joel Svensson*
Chalmers University of Technology
Gothenburg, Sweden
VESC Labs AB
Sexdrega, Sweden
bo.joel.svensson@gmail.com

Benjamin Vedder
VESC Labs AB
Sexdrega, Sweden
Lind Art and Technology
Stockholm, Sweden

arXiv:2609.23009v1 [cs.PL] 19 Sep 2026

Abstract

LispBM is a Lisp dialect for microcontrollers and embedded systems. This paper is about the changes made to LispBM resulting from it being integrated into the VESC family of firmware and thus gaining a userbase.

As a surprise to Author 1, the new users of LispBM wrote programs larger than ever imagined. This used up all available heap space and we were forced to find a way to evaluate Lisp programs partitioned across RAM and flash memory. This story and a couple more, like the one about the Lisp-image system for fast boots, are the core contribution of this paper.

When the first versions of the VESC firmware with a built-in Lisp runtime system were released, there was some grumbling here and there about Lisp and its syntax, as expected. To our delight, most seemed very excited about a scripting language in the firmware. To the engineer and builder users, the choice of syntax seemed to matter less than the potential it opened up for them.

CCS Concepts

• **Software and its engineering** → **Interpreters**; **Memory management**; **Scripting languages**; **Concurrent programming languages**.

Keywords

LispBM, Lisp, embedded systems, microcontrollers, scripting languages, garbage collection, concurrent programming, language design

ACM Reference Format:

Bo Joel Svensson and Benjamin Vedder. 2027. A Linked List of Cases in Language Design. In *Proceedings of 38th Symposium on Implementation and Application of Functional Languages (IFL 2026).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

LispBM is a dialect in the Lisp family of languages designed from the beginning with microcontrollers as the primary target platform.

*Docent, unsalaried.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
IFL 2026, TBD
© 2027 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2026/TBD
https://doi.org/XXXXXXX.XXXXXXX

Early in the development of LispBM, design choices were driven by the 1st author's interests and curiosity, first as a way to side-step the time-consuming compile-flash-debug cycle of embedded systems development. Later, while collaborating with Krook on Scoria [12], a language for real-time IoT applications, and Sarkar on Synchron [15, 16], an API and runtime for embedded systems, experience gained from building and running LispBM fed directly into that work. The kinship with Synchron runs deeper than a shared target platform: both systems ended up with similar garbage collectors and heap designs. LispBM and Synchron chose different paths on concurrency, however: Sarkar chose synchronous message passing à la Concurrent ML. For LispBM we chose asynchronous message passing with a pattern matching receive inspired by Erlang [2].

In December of 2022 we started integrating LispBM into the VESC (Vedder Electronic Speed Controller) motor control firmware. At the time, author 1 expected this to be a short-lived experiment. LispBM, however, has remained a part of the VESC firmware, and is by now deployed in hundreds, if not thousands, of electric vehicles ranging from hobbyist machines to luxury range electric surfboards. With this change in scope, LispBM was thrown into a situation where there are users, requirements and drivers other than scientific curiosity.

This paper is about the choices, changes and problems that occur when a particular language implementation project gains a userbase and a whole new set of requirements compared to the premises it was built around before. Here we present our experience in steering LispBM to be the best language we can make it for its new role as a scripting language embedded into the VESC family of firmware.

The next section, Section 2, is a primer on what LispBM programs and implementation look like. The sections that follow the primer each tell a story of how the LispBM language was changed, improved, or just pragmatically steered to align with its new direction.

2 LispBM: A Primer

The initial motivator behind LispBM was to get a read-evaluate-print-loop (REPL) onto a microcontroller and in the process learn about both Lisp implementation and microcontroller limitations. LispBM does not attempt to be a Common Lisp [9] or a Scheme [19] or compatible with any other pre-existing language in the Lisp family [13], rather we chose to be free to implement as we see fit in relation to our needs and capabilities and the limitations of the target platform.

LispBM is a portable Lisp interpreter that you can slot into your firmware on any platform that has a C compiler. Indeed, it is a conscious choice not to include any platform-specific code into

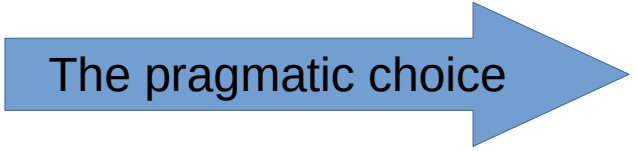
https://arxiv.org/pdf/2609.23009

Concurrency model

- Old
  - Cooperative.
  - Interleavings are predictable.
  - Potentially starves.
  - Cognitive load on programmer.
- New
  - Pre-emptive round robin.
  - Based on time (2025).
  - Easier for novice.
  - Introduces races.
  - More complicated scheduler.

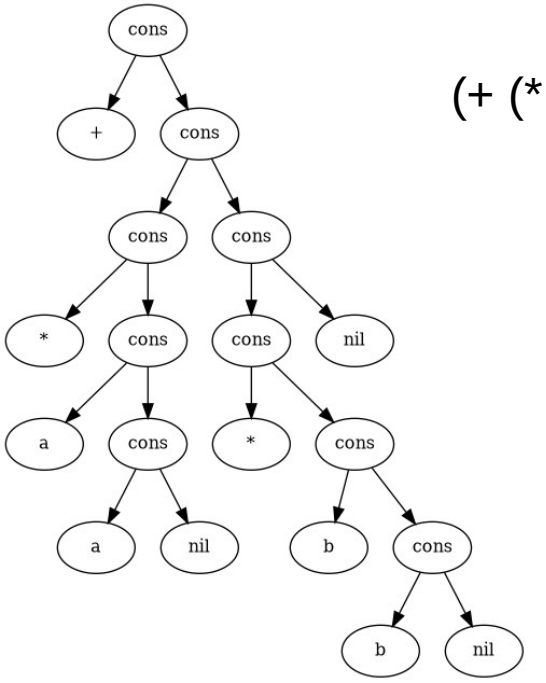
Concurrency model

- Old
  - Cooperative.
  - Interleavings are predictable.
  - Potentially starves.
  - Cognitive load on programmer.
- New
  - Pre-emptive round robin.
  - Based on time (2025).
  - Easier for novice.
  - Introduces races.
  - More complicated scheduler.



Programs larger than RAM

- The memory system
  - Heap of cons cells
    - From ~2000 cells and up.
  - Arrays memory
    - Larger
    - Stores boxed values, byte arrays, thread contexts etc.



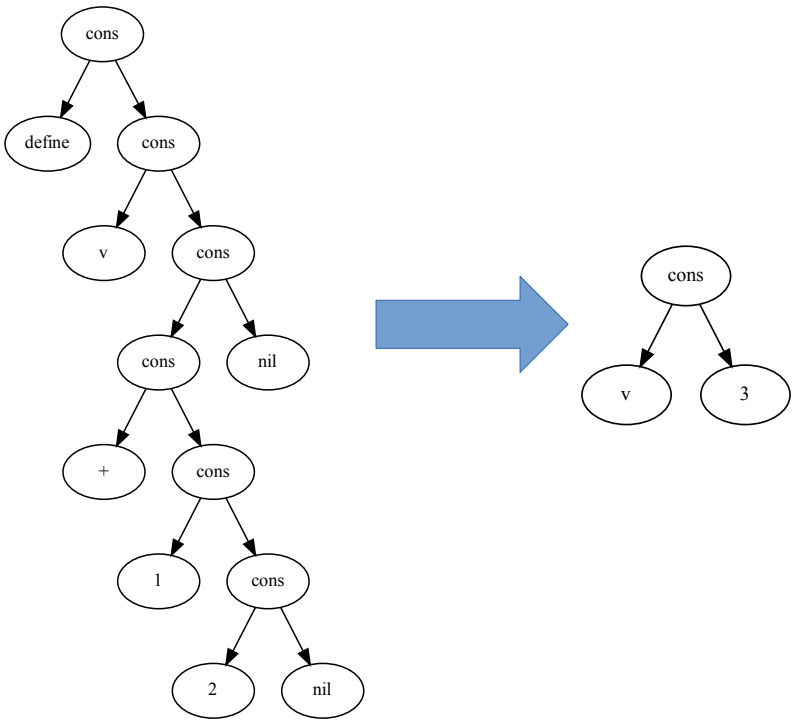
(+ (* a a) (* b b))

Programs larger than RAM

- The incremental reader
  - Interleave read with evaluate.
- The const-heap concept
  - Stored in flash
- Introduces a “flash footprint” determinism requirement on programs.

- Flash
  - Write seldom memory.
  - Byte writable. (a cleared byte can be written to)
  - Segment clearable. Can be 4K can be 128K.
  - Limited number of erase cycles.

Programs larger than RAM



- Incremental read
  - The program can be reclaimed.
  - Heap requirement is just the environment binding.
- If “const-heap” the value is stored in flash (if composite or array).

Programs larger than RAM

```
@const-start

(define a (list 1 2 (systime)))

@const-end


(print a)
```

- Defines in a “const-block” are created on the flash heap.
  - Can not be mutated.

Fast boot and Lisp images

- Up until now each boot re-reads the source code.
  - Each top level expression is evaluated once read.
  - Definitions within const-blocks are stored in flash.
    - On a reboot this requires that the exact same bytes are present in flash.

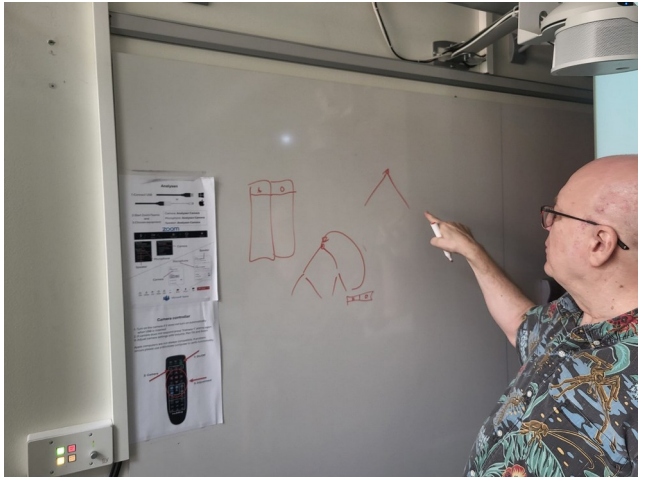


Fast boot and Lisp images

- Boot from a saved image.
- What goes into the image
  - The const-heap is already in flash.
  - Environment bindings are serialised (flattened) and stored in flash.

A new puzzle

- Restoring from an image must restore sharing.



Fast boot and Lisp images

- Bonus
  - Source is evaluated only once. So the reader flash-determinism requirement is less of a problem.

Fast boot and Lisp images

- Not stored in image:
  - Thread state.
  - Opened resources.
- Instead:
  - A “main” function is required and runs as part of image restore.

Fast boot and Lisp images

```
@const-start

(define a ...)
(define b ...)

(defun main () {
       (init-1)
       (init-2)
       (spawn thread1)
       (spawn thread2)
       })
@const-end

(image-save)
(main)
```

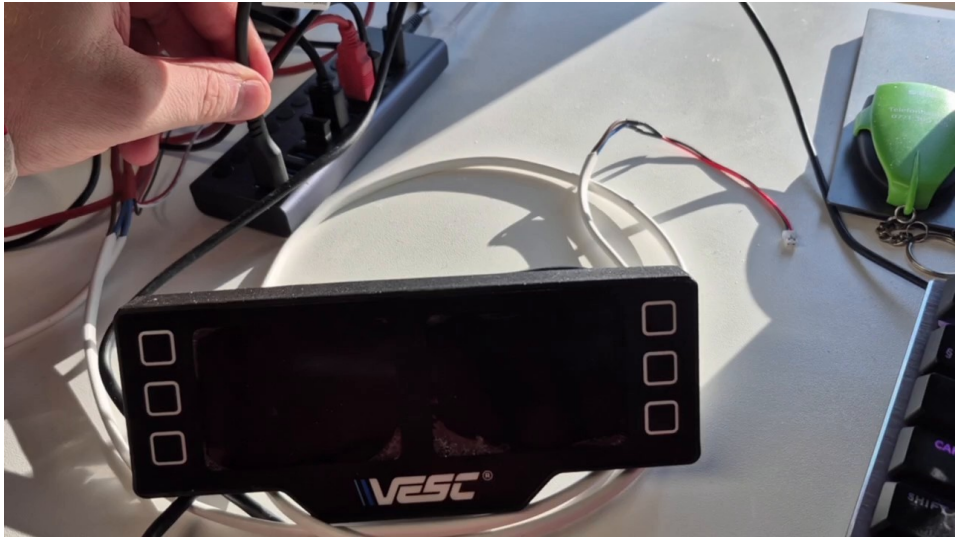
Thank you!

- Question?

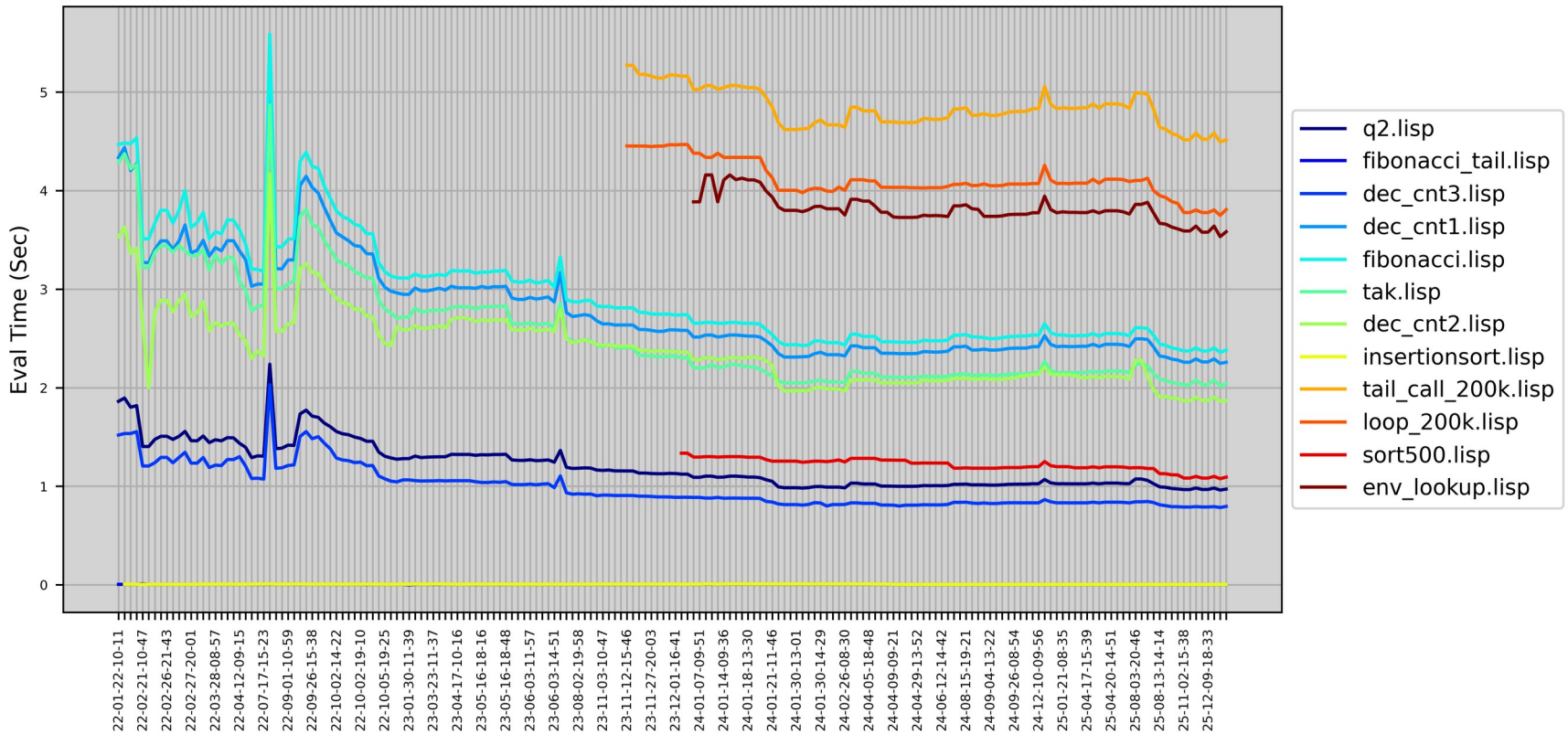
- Thanks to:
  - The community of contributors and users.
  - The VESC project.
  - Lind Art and Technology.

Extra slides


VESC®

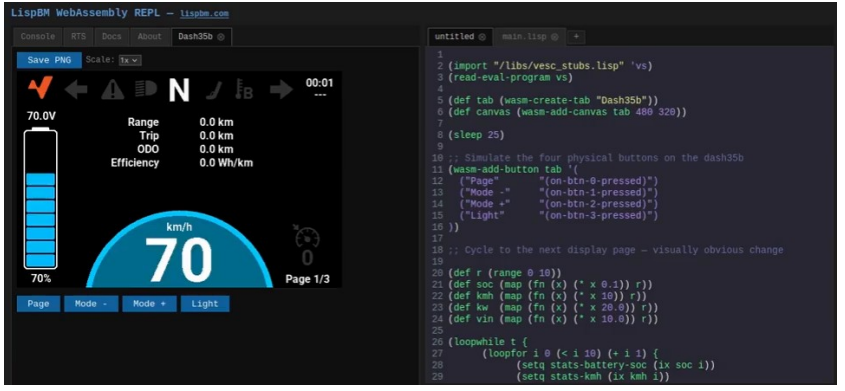

VESC®

Performance over time



Compacting allocator

- Motivation
  - Guarantee there is room for the set of screen buffers needed in GUI applications.
- Memories
  - Heap of cons cells is immune to fragmentation.
  - Arrays memory is for arbitrary size allocations and suffers from fragmentation.
  - Compactable memory areas are allocated from the “arrays memory”.



Cooperative vs pre-emptive

```
(defun spin (k)
    (loopwhile (> k 0) (setq k (- k
1))))

(defun worker (name)
  (loopwhile t
             (progn (print name)
                    (spin 100000))))

(spawn worker "A")
(spawn worker "B")
```

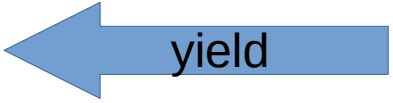
- Cooperative: A A A A A A
- Pre-emptive: A B A B A B A B

Cooperative vs pre-emptive

```
(defun spin (k)
    (loopwhile (> k 0) (setq k (- k
1))))

(defun worker (name)
  (loopwhile t
             (progn (print name)
                    (spin 100000))))

(spawn worker "A")
(spawn worker "B")
```



Cooperative vs pre-emptive

```
(define counter 0)

(defun inc (n)
  (loopwhile (> n 0)
      (progn (setq counter (+ counter
1))
             (setq n (- n 1)))))

(spawn inc 100000)
(spawn inc 100000)
```

- Cooperative: counter = 200000
- Pre-emptive: counter = 179907 (for example)

Cooperative vs pre-emptive

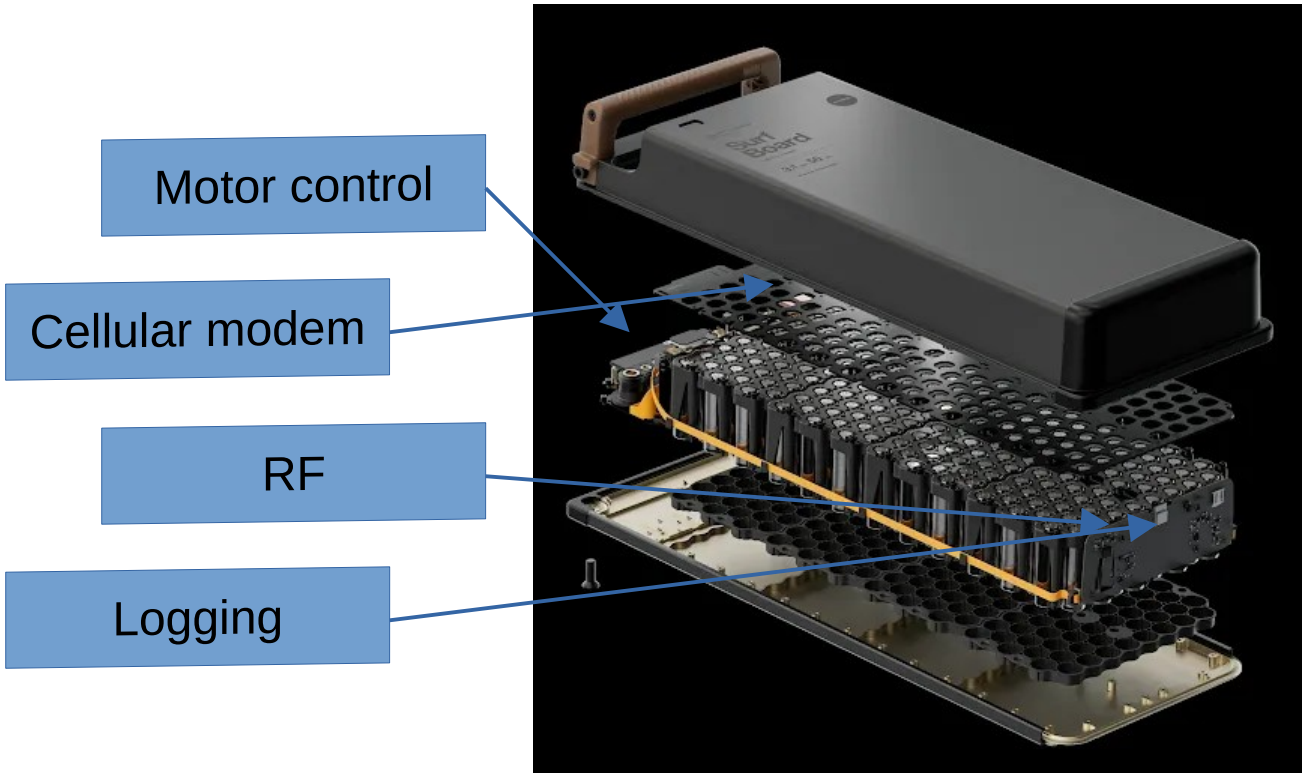
```
(define counter 0)

(defun inc (n)
  (loopwhile (> n 0)
      (atomic (setq counter (+ counter
1))
              (setq n (- n 1)))))

(spawn inc 100000)
(spawn inc 100000)
```

Surfboard distributed computing

- 5 MCUs
- CAN bus





- Wheelie mode being developed.

- Contributions to graphics libraries have increased recently.



ESP32-C3 (RISC-V No hard FP)