LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Kludging The editor with The compiler

Andrea Corallo
<akrl@sdf.org>

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

## What GNU Emacs is ?

- It's a Lisp implementation (Emacs Lisp)
- Its task is to slurp unstructured text from the OS
- Lisp programs can represent manipulate and share these data

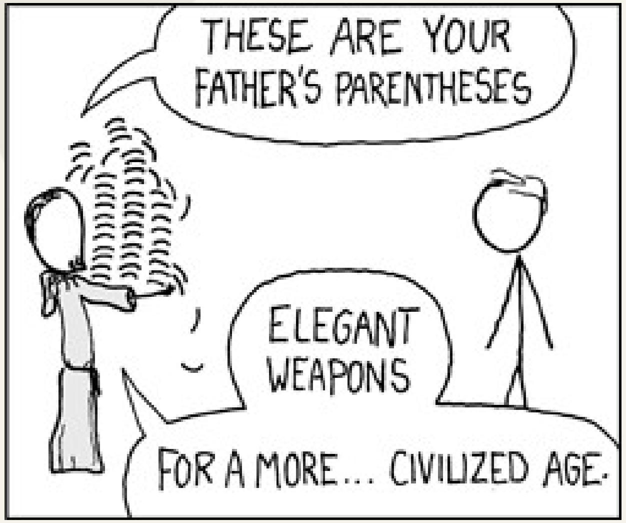
LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

## Emacs Lisp

- Continuously improving
- Capable of (almost) any task
- Surprisingly spread
  - Emacs 1.794.561 LOC
  - emacsmirror.net 9.888.547 LOC!!
  - 21th in push number (https://madnight.github.io)

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Lisp


THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

xkcd.com

- Dynamic
- Homoiconic
- Maxwell’s equations of software (Alan Kay)

- Easy to learn... easy to implement!

~1500

© The Scream 1895/Edvard Munch

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

## Implementation

- ~30% is C
- Lisp Interpreted or byte-compiled
- Byte compiler is written in Elisp
- Must bootstrap!
- Byte-code runs on a stack-based VM

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

## Where to improve

- Namespace
- Extensibility
- Performance
  - Garbage Collector
  - Execution Engine
  - Real multi-threading
- Debuggability and compile time errors

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

# Where to improve

- Namespace
- Extensibility
- Performance
  - Garbage Collector
  - Execution Engine
  - Real multi-threading
- Debuggability and compile time errors

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

## Where to improve

- Namespace
- Extensibility
- Performance
  - Garbage Collector
  - Execution Engine
  - Real multi-threading
- Debuggability and compile time errors

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

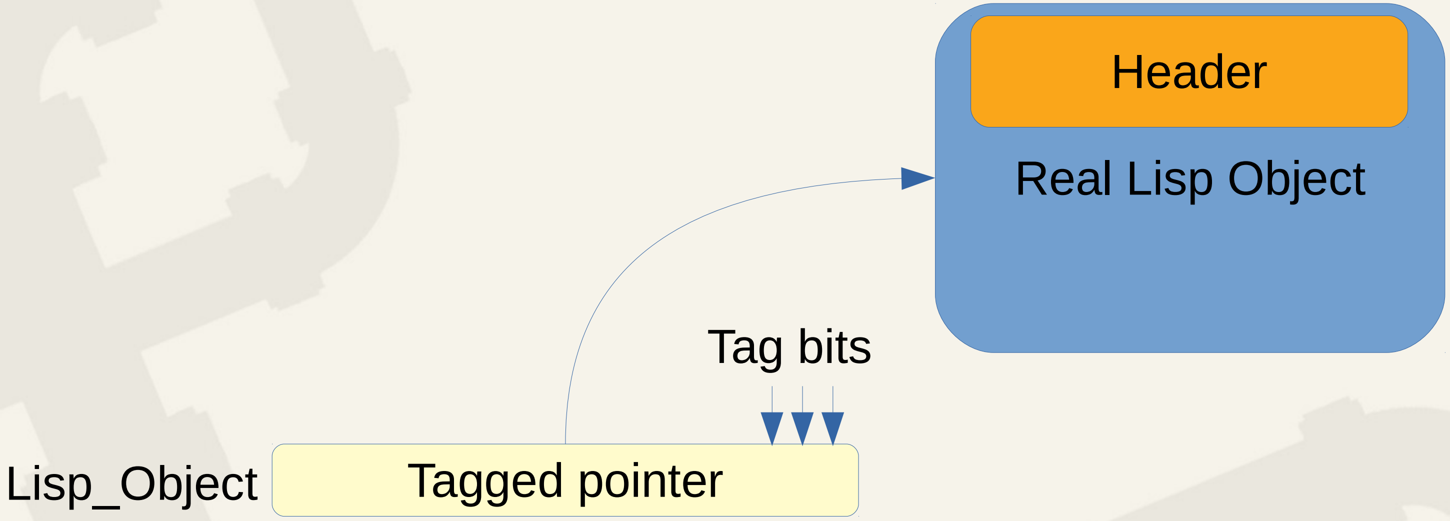
## How to improve

Improving the Lisp performance allow for:

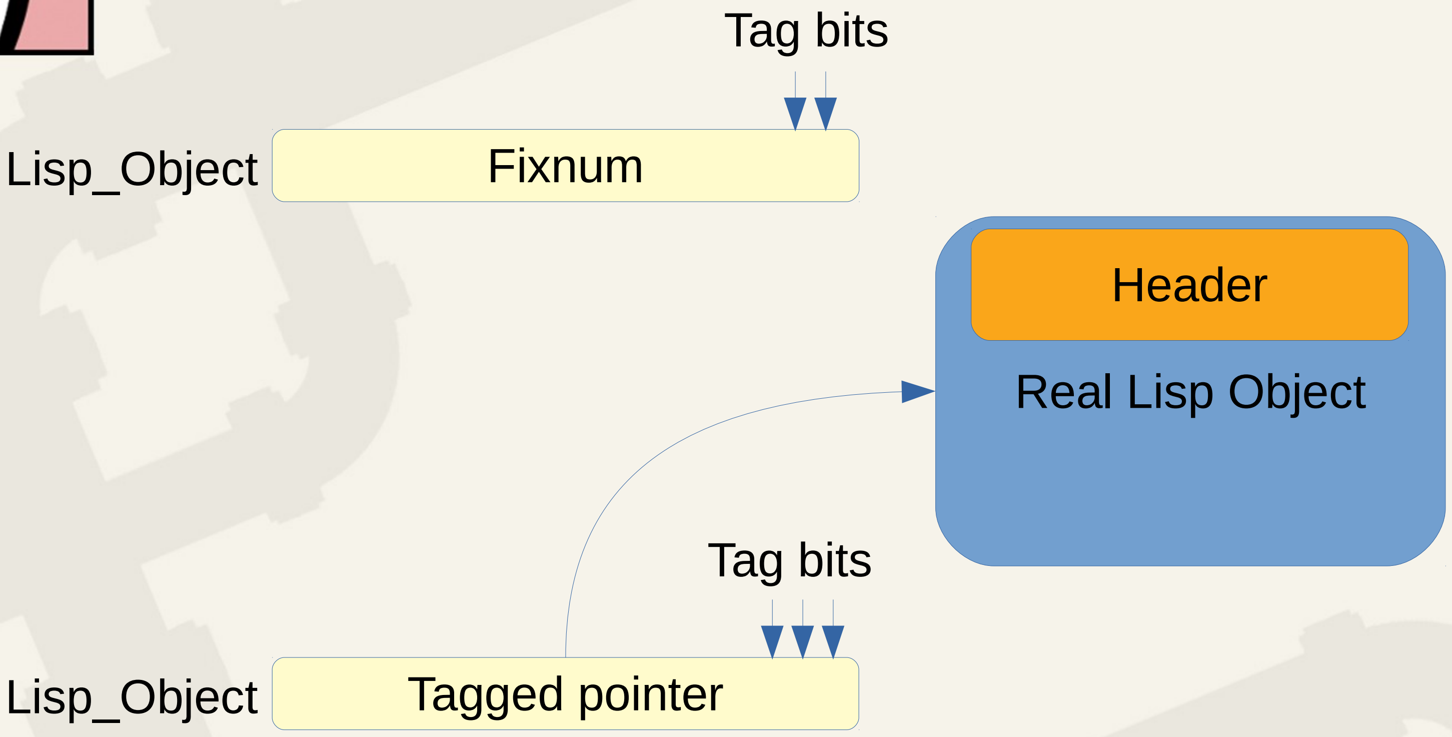
- Less C to be written and maintained
- Ease write of performance critical extensions

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Lisp Objects



# Lisp Objects



## Elisp VM

A stack base push and pop VM

- Lisp

```
(* (+ a 2) 3)
```

- LAP

```
(byte-varref a)
(byte-constant 2)
(byte-plus)
(byte-constant 3)
(byte-mult)
(byte-return)
```

# Elisp VM

Exec stack
SP

- LAP

  (byte-varref a)

  (byte-constant 2)

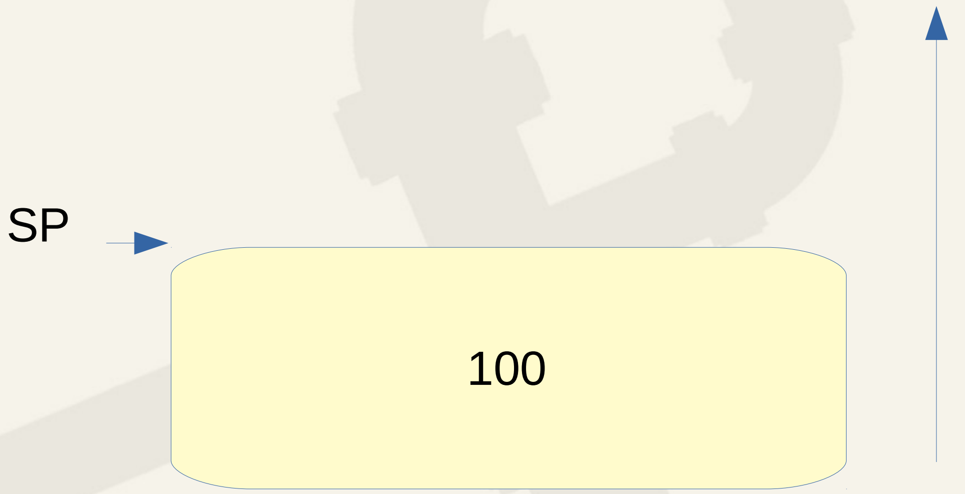
  (byte-plus)

  (byte-constant 3)

  (byte-mult)

  (byte-return)

## Elisp VM

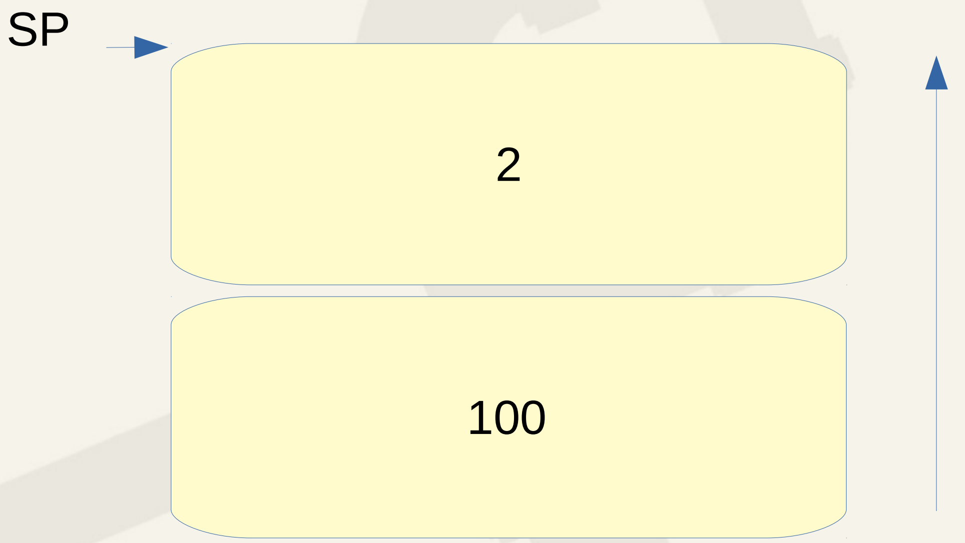


Exec stack

- LAP

```
(byte-varref a) <=
(byte-constant 2)
(byte-plus)
(byte-constant 3)
(byte-mult)
(byte-return)
```

# Elisp VM



Exec stack

- LAP

```
(byte-varref a)
(byte-constant 2) <=
(byte-plus)
(byte-constant 3)
(byte-mult)
(byte-return)
```

# Elisp VM

- LAP

(byte-varref a)

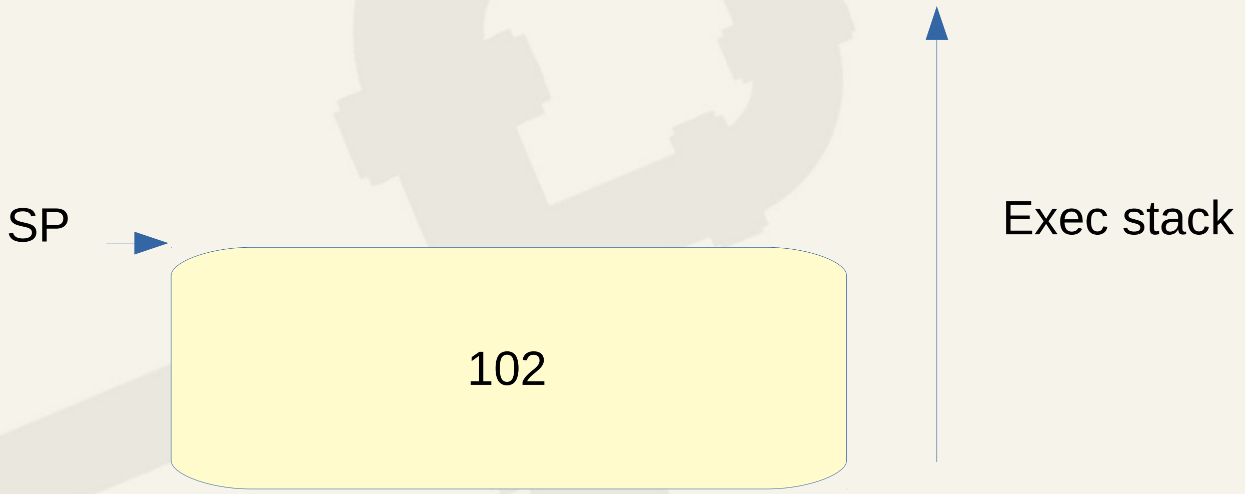
(byte-constant 2)

(byte-plus) <=

(byte-constant 3)

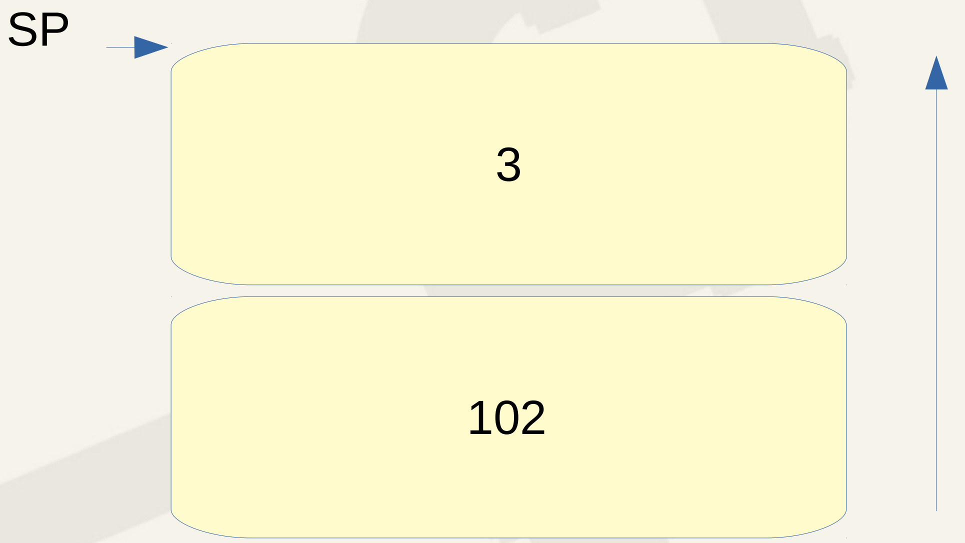
(byte-mult)

(byte-return)



LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Elisp VM



Exec stack

- LAP

(byte-varref a)

(byte-constant 2)

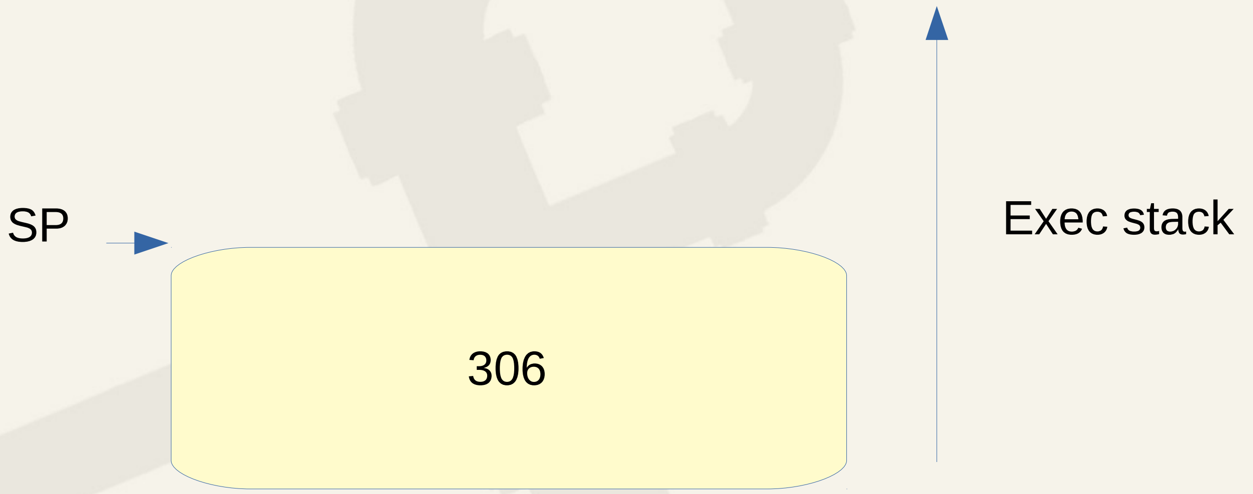
(byte-plus)

(byte-constant 3) <=

(byte-mult)

(byte-return)

# Elisp VM



- LAP

```
(byte-varref a)
(byte-constant 2)
(byte-plus)
(byte-constant 3)
(byte-mult) <=
(byte-return)
```

# Elisp VM

Exec stack
SP

- LAP

  (byte-varref a)

  (byte-constant 2)

  (byte-plus)

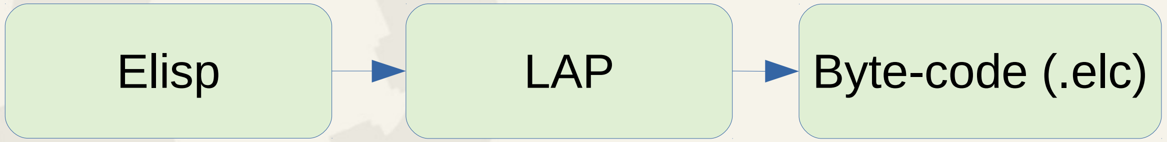
  (byte-constant 3)

  (byte-mult)

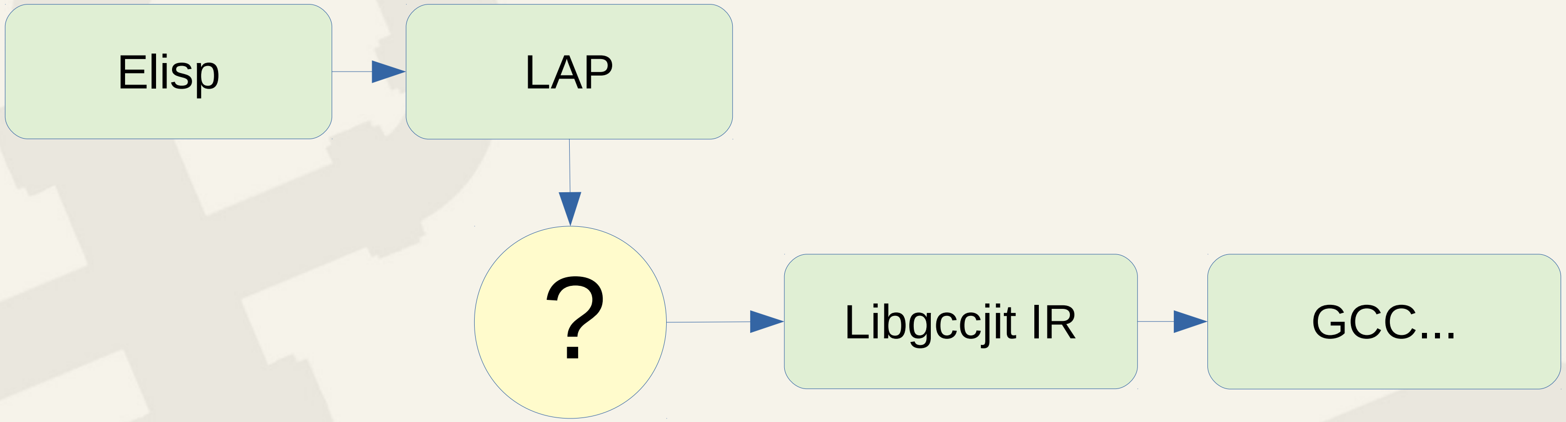
  (byte-return) <=

# Byte-compiler pipeline

- Macro expansion
- Closure conversion
- Source level optimizations
- Single pass byte-compiler => LAP (Lisp Assembly Program)
- Peephole LAP optimizations
- Assembled into byte-code


Elisp
LAP
Byte-code (.elc)

LINUX PLUMBERS CONFERENCE
August 24-28, 2020



LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

# libgccjit

- Added by David Malcolm in GCC 5
- Describe programmatically a C-ish semantic
- Good for Jitters or AoT compilers

# A simple translation

- LAP

```
(byte-varref a)
(byte-constant 2)
(byte-plus)
(byte-constant 3)
(byte-mult)
(byte-return)
```

- C

```
Lisp_Object local[2];
local[0] = varref (a);
local[1] = two;
local[0] = plus (local[0], local[1]);
local[1] = three;
local[0] = mult (local[0], local[1]);
return local[0];
```

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

## Optimizing outside GCC

- Generate code effectively optimizable
- Provide user feedback
  - warning
  - errors
  - optimizations hints

## Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Lisp

C

# Native compiler pipeline

| | |
|---|---|
| • spill-lap | LAP |
| • limplify | |
| • fwprop | |
| • call-optim | |
| • ipa-pure | |
| • fwprop | LIMPLE |
| • dead-code | |
| • tco | |
| • fwprop | |
| • remove-type-hints | |
| • final | Libgccjit IR |

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Native compiler pipeline

- spill-lap ⟵
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Run the byte-compiler infrastructure to obtain LAP

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Native compiler pipeline

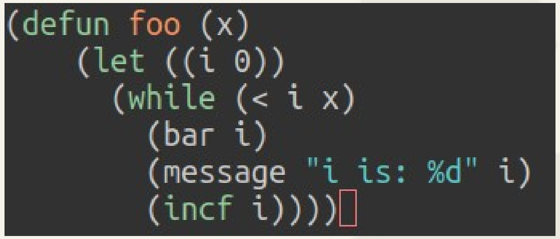
- spill-lap
- limplify ←
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Convert LAP into LIMPLE

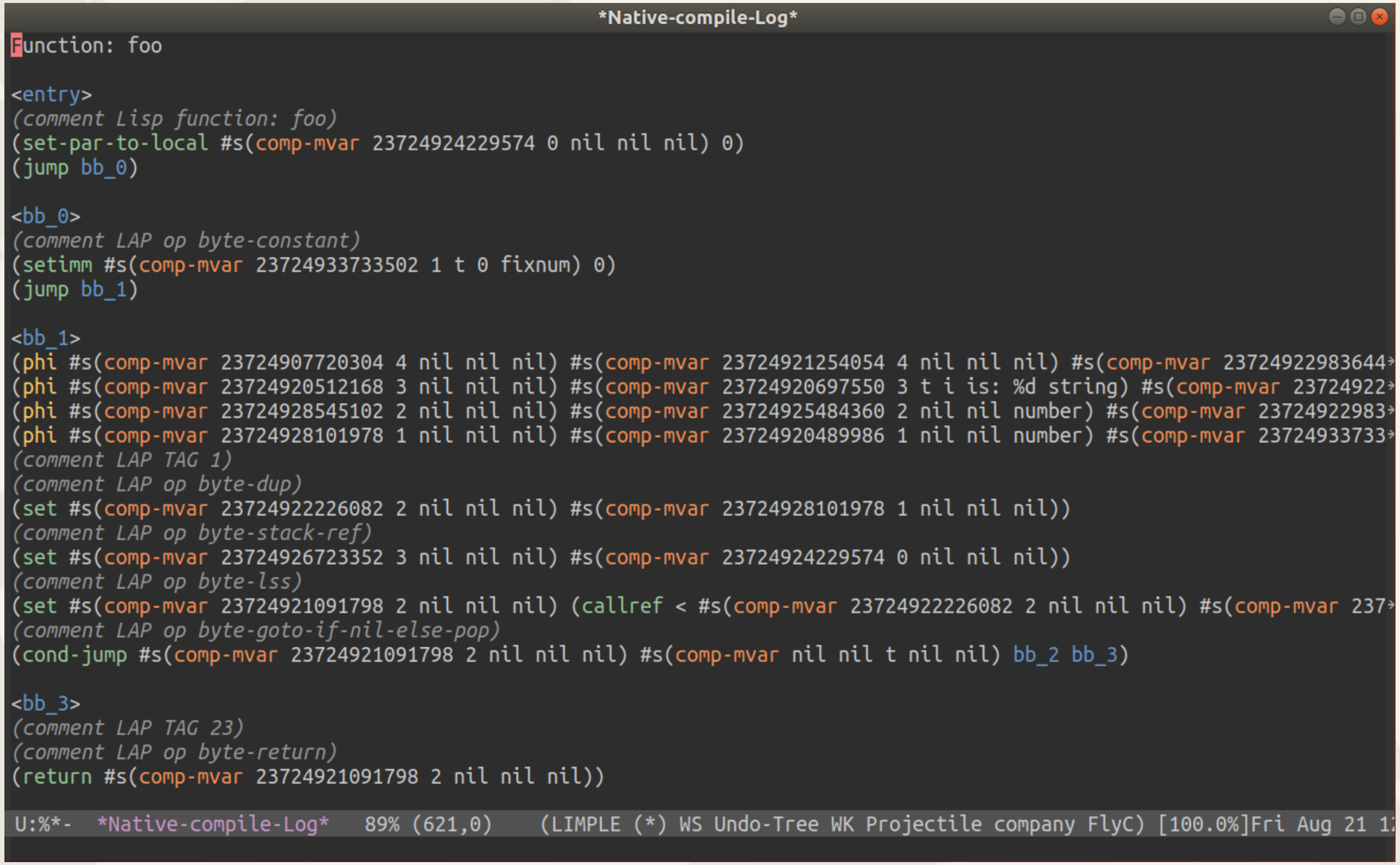
- LIMPLE as tribute to GIMPLE
- CFG based
- SSA

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

# LIMPLE



```
(defun foo (x)
    (let ((i 0))
      (while (< i x)
        (bar i)
        (message "i is: %d" i)
        (incf i))))
```



*Native-compile-Log*

```
Function: foo

<entry>
(comment Lisp function: foo)
(set-par-to-local #s(comp-mvar 23724924229574 0 nil nil nil) 0)
(jump bb_0)

<bb_0>
(comment LAP op byte-constant)
(setimm #s(comp-mvar 23724933733502 1 t 0 fixnum) 0)
(jump bb_1)

<bb_1>
(phi #s(comp-mvar 23724907720304 4 nil nil nil) #s(comp-mvar 23724921254054 4 nil nil nil) #s(comp-mvar 23724922983644
(phi #s(comp-mvar 23724920512168 3 nil nil nil) #s(comp-mvar 23724920697550 3 t i is: %d string) #s(comp-mvar 23724922
(phi #s(comp-mvar 23724928545102 2 nil nil nil) #s(comp-mvar 23724925484360 2 nil nil number) #s(comp-mvar 23724922983
(phi #s(comp-mvar 23724928101978 1 nil nil nil) #s(comp-mvar 23724920489986 1 nil nil number) #s(comp-mvar 23724933733
(comment LAP TAG 1)
(comment LAP op byte-dup)
(set #s(comp-mvar 23724922226082 2 nil nil nil) #s(comp-mvar 23724928101978 1 nil nil nil))
(comment LAP op byte-stack-ref)
(set #s(comp-mvar 23724926723352 3 nil nil nil) #s(comp-mvar 23724924229574 0 nil nil nil))
(comment LAP op byte-lss)
(set #s(comp-mvar 23724921091798 2 nil nil nil) (callref < #s(comp-mvar 23724922226082 2 nil nil nil) #s(comp-mvar 237
(comment LAP op byte-goto-if-nil-else-pop)
(cond-jump #s(comp-mvar 23724921091798 2 nil nil nil) #s(comp-mvar nil nil t nil nil) bb_2 bb_3)

<bb_3>
(comment LAP TAG 23)
(comment LAP op byte-return)
(return #s(comp-mvar 23724921091798 2 nil nil nil))

U:%*-  *Native-compile-Log*   89% (621,0)    (LIMPLE (*) WS Undo-Tree WK Projectile company FlyC) [100.0%]Fri Aug 21 1
```

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

## Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

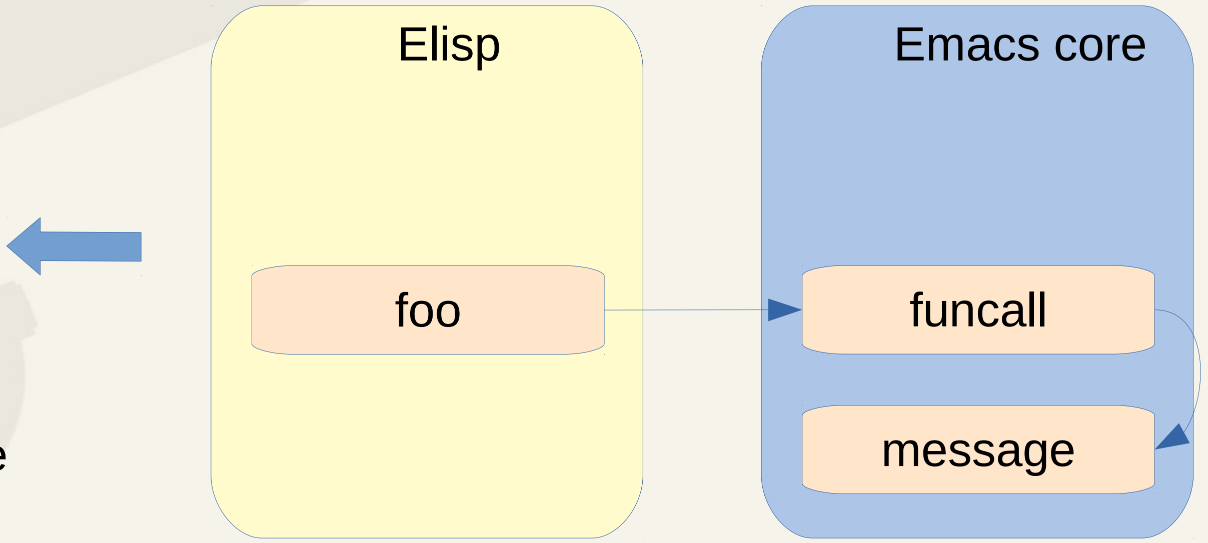
– Forward propagate types and values

– Execute in the run-time pure functions

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

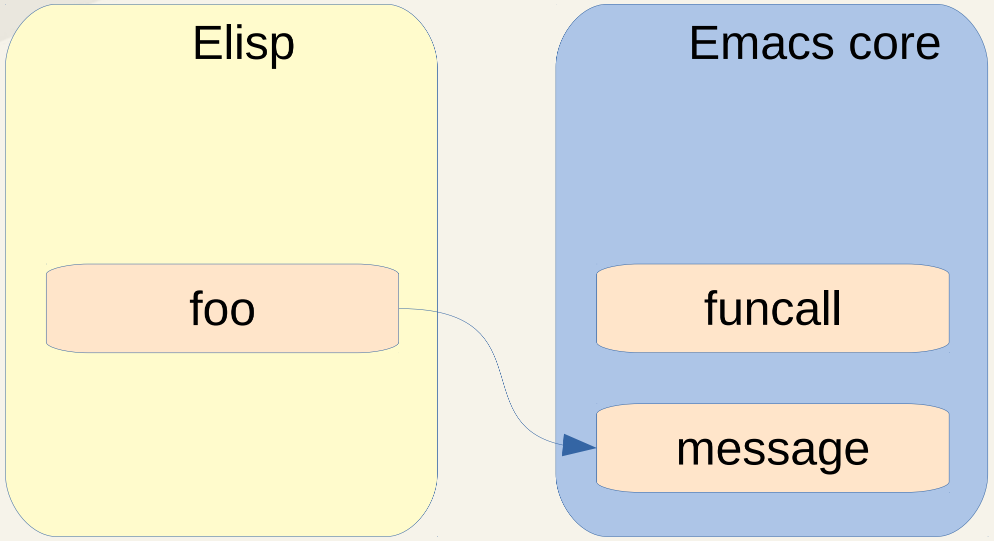
# Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final



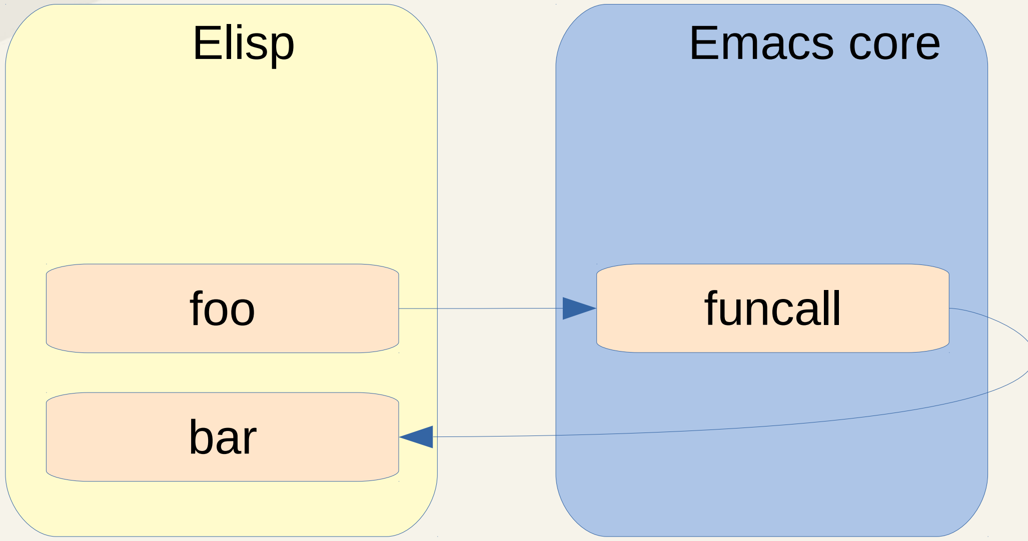
# Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final



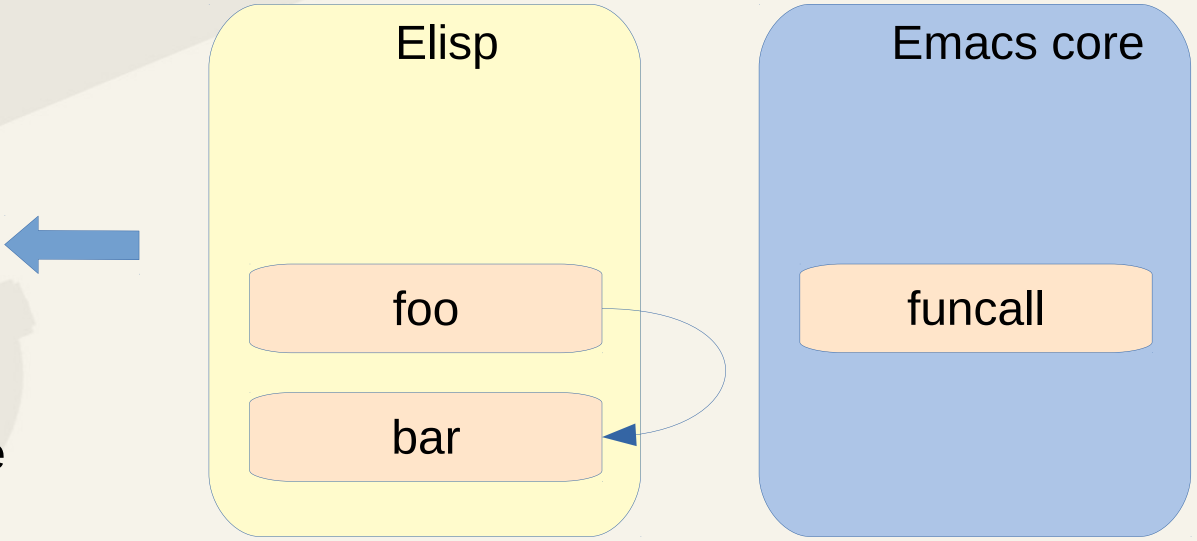
# Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final



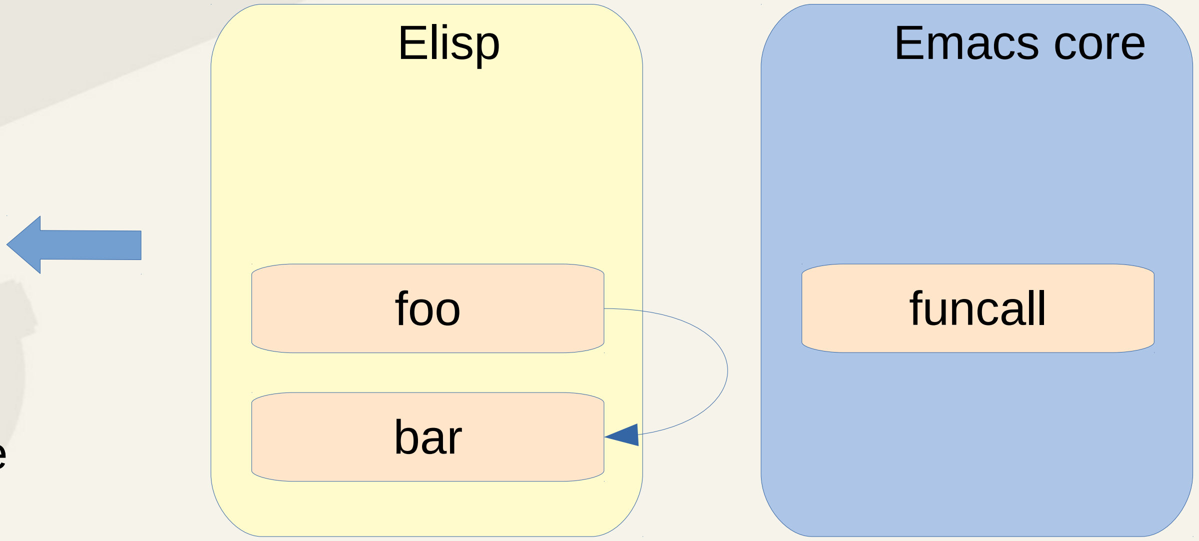
# Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final



# Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final



Allow GCC IPA logic

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure ←
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

– Infer function purity

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

## Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco ←
- fwprop
- remove-type-hints
- final

Tail Recursion Elimination

– Pattern match and replace recursive calls in tail position

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

## Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final ←

Convert LIMPLE into libgccjit IR

– Define inline functions to access fundamental data types

– Use type information for code generation

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Extending the language

- speed [0-3] ←
- Compilation unit
- Compiler hints

Borrowed from CL

– Allow for some cheating at 3

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

# Extending the language

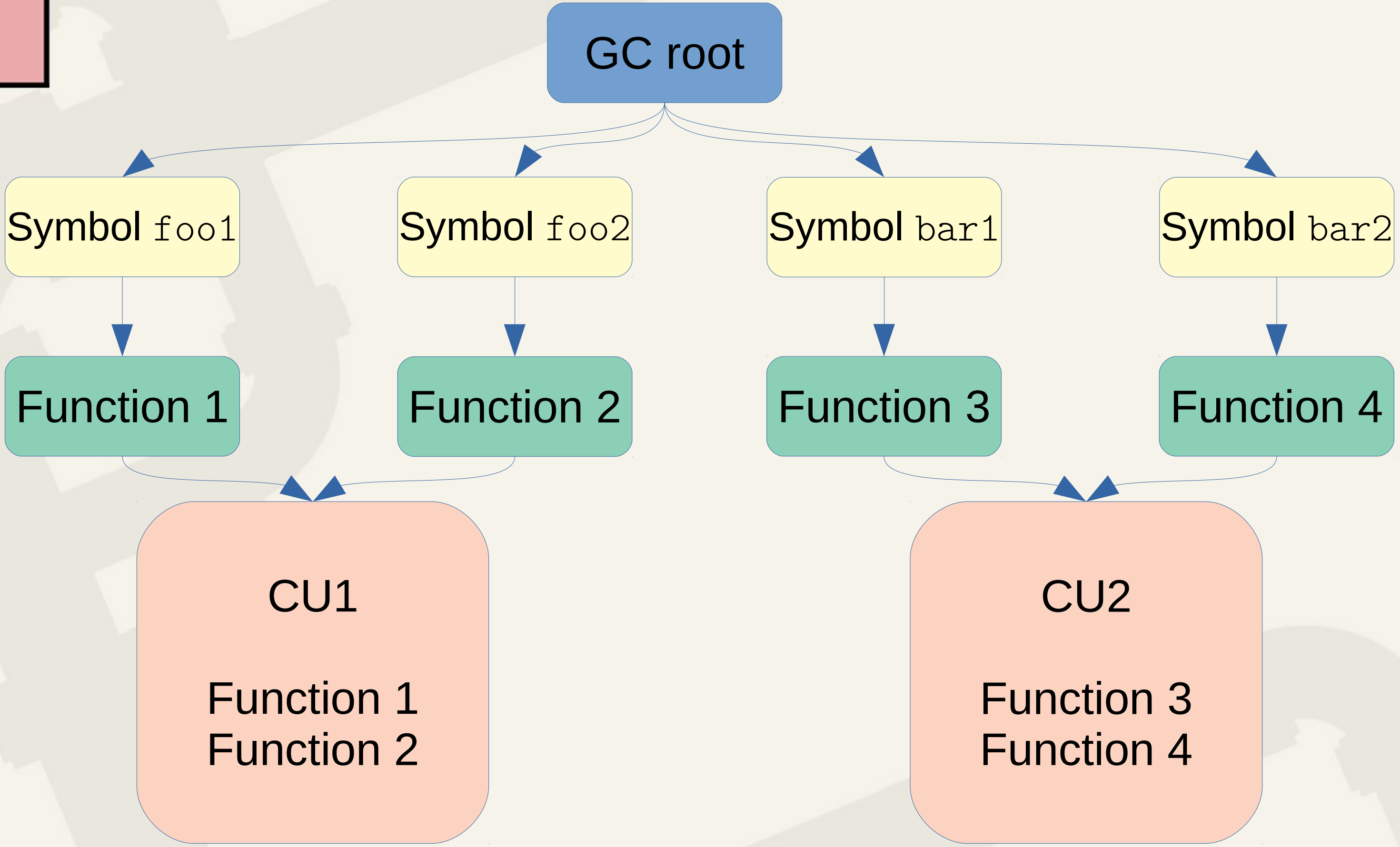
- speed [0-3]
- Compilation unit ←
- Compiler hints

First class object

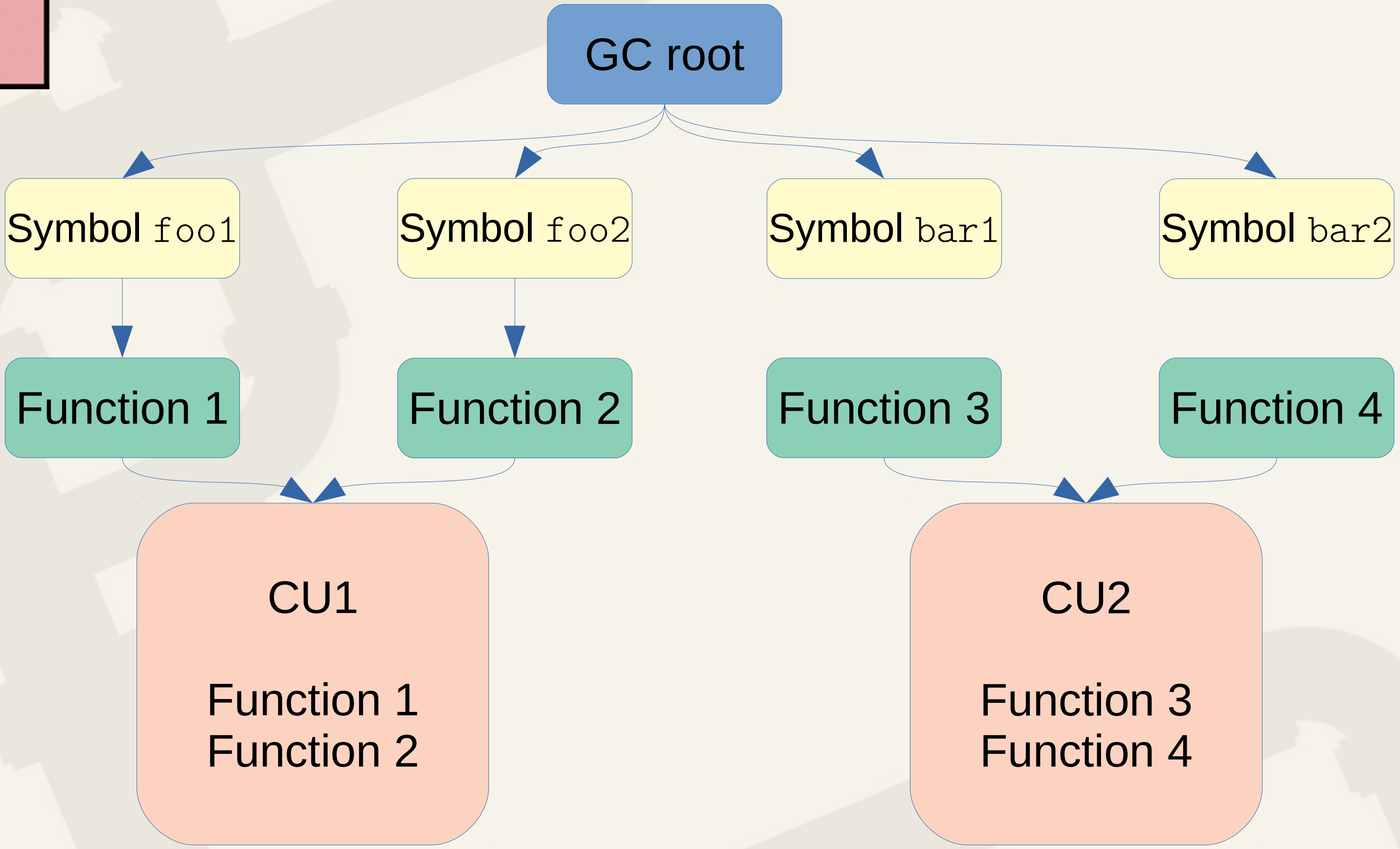
– Allow for the GC to handle loaded functions

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

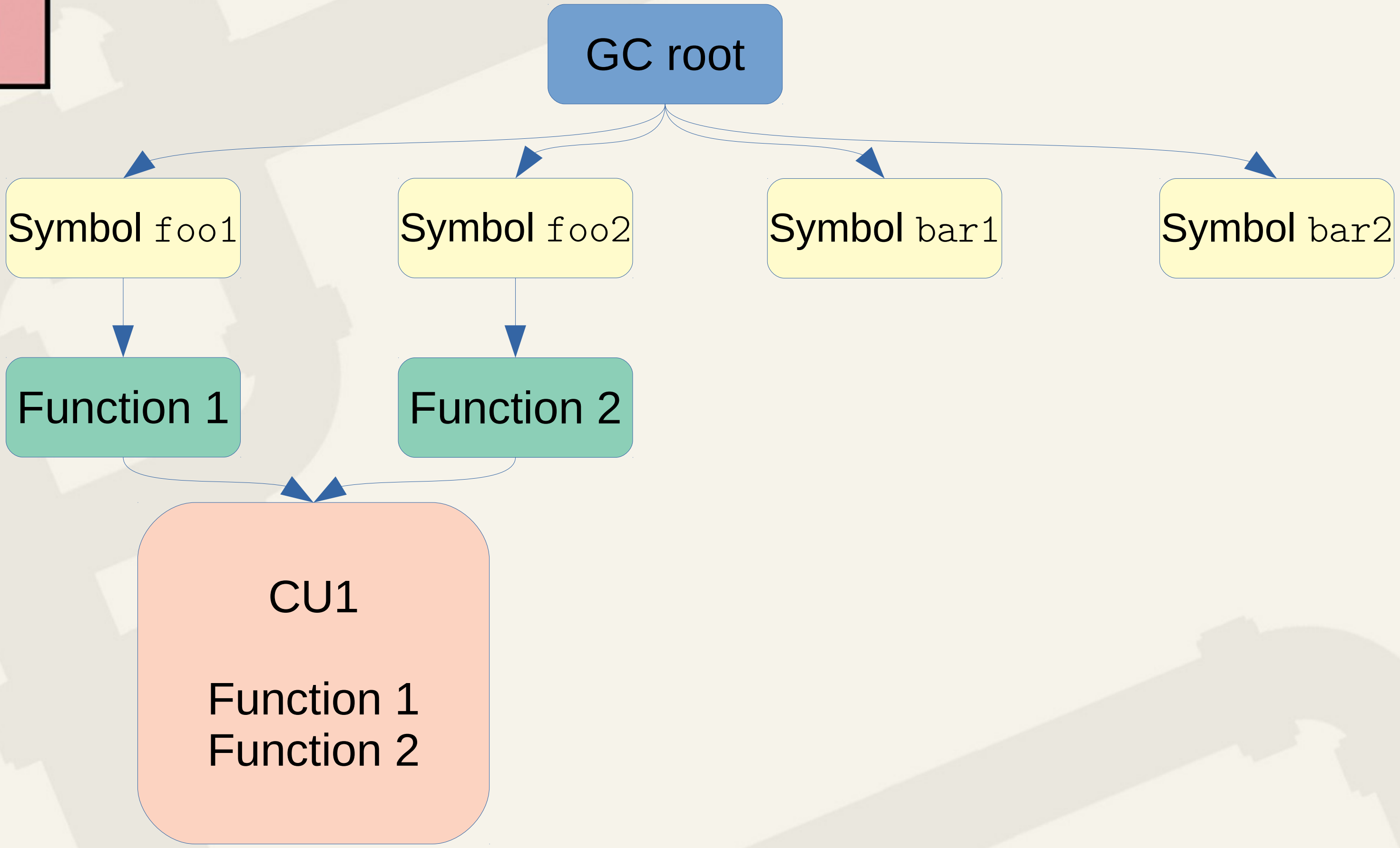
# Compilation Unit



# Compilation Unit



# Compilation Unit



LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Extending the language

- speed [0-3]
- Compilation unit
- Compiler hints

Suggest a type for an expression

– Assertion or hint for removing the type check

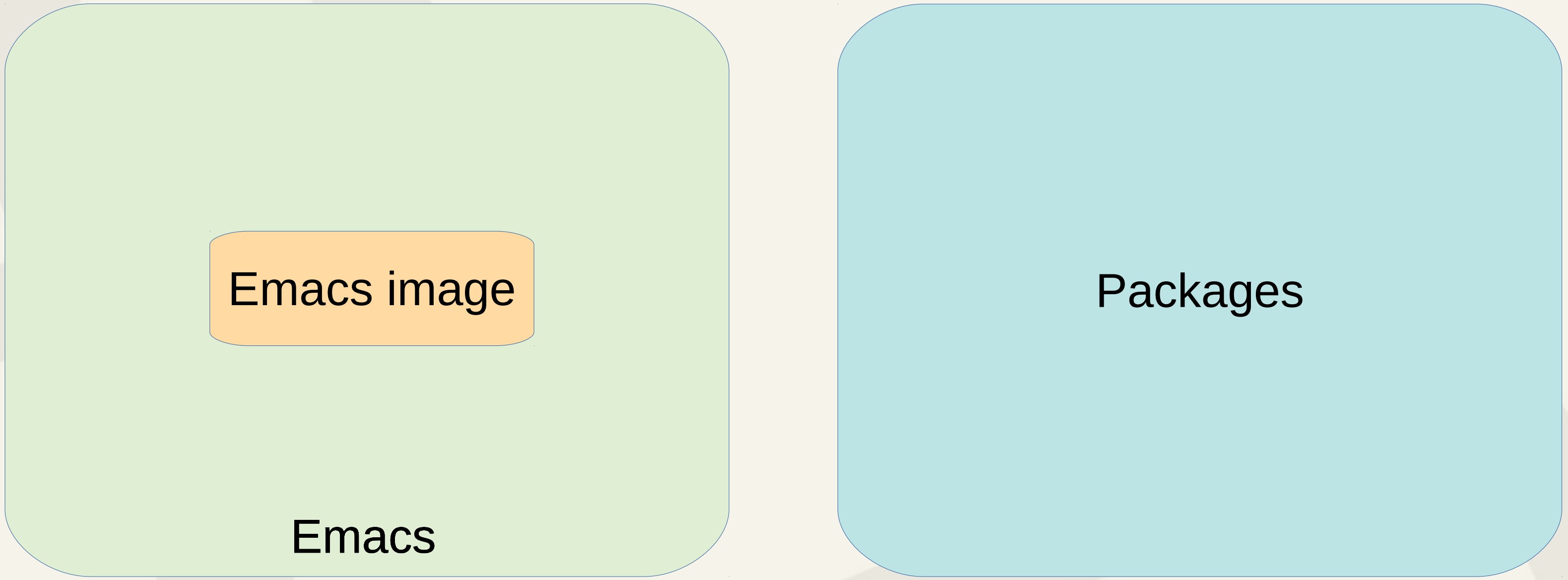
```
(setf x (1+ x))

(setf x (comp-hint-fixnum (1+ x)))
```

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

## Jit vs AoT

- Born as an Ahead of Time compiler
- Moved to an Hybrid approach



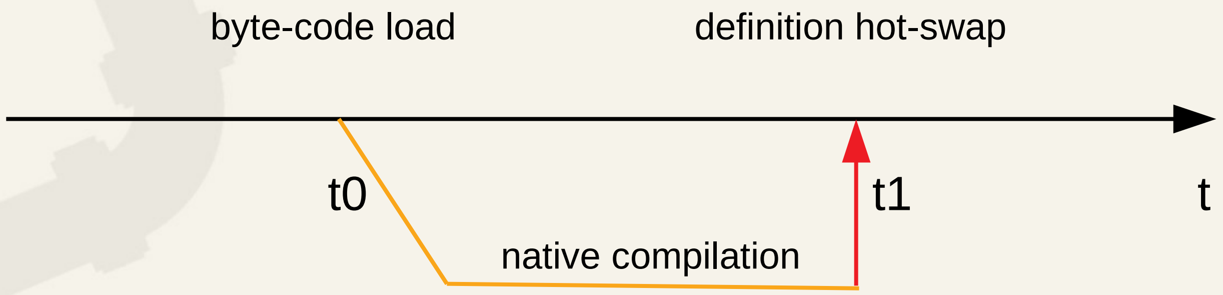
LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

## Async compilation

- Jit like triggered
- Parallel
- Output reused between different sessions
- Definitions hot-swap

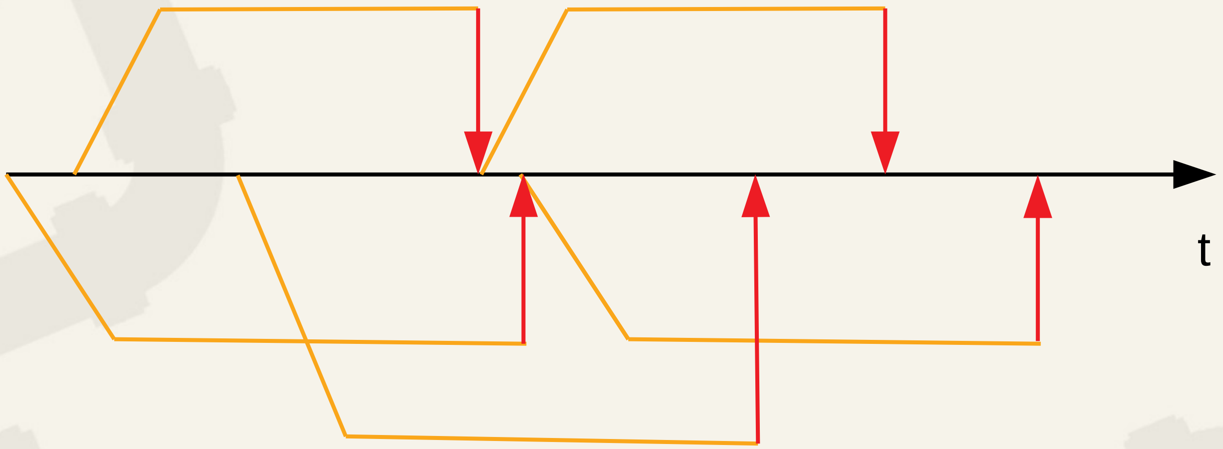
LINUX PLUMBERS CONFERENCE
August 24-28, 2020

# Async compilation



LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

# Async compilation



LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

## Performance

```
| test            | runtime (s) | runtime (s) | perf uplift |
|-----------------+-------------+-------------+-------------|
| fibn-rec        |        7.24 |        0.00 |         --- |
| fibn-tc         |        6.40 |        0.01 |         --- |
| fibn            |       11.91 |        0.00 |         --- |
| listlen-tc      |        8.81 |        0.21 |       42.0x |
| nbody           |       17.10 |        2.41 |        7.1x |
| inclist         |       15.01 |        2.39 |        6.3x |
| bubble          |       12.13 |        2.55 |        4.8x |
| pcase           |       11.93 |        2.70 |        4.4x |
| flet            |       12.15 |        3.58 |        3.4x |
| bubble-no-cons  |       10.81 |        3.64 |        3.0x |
| dhrystone       |        8.78 |        3.63 |        2.4x |
| pidigits        |       14.51 |        9.47 |        1.5x |
| map-closure     |        9.33 |        9.32 |        1.0x |
|-----------------+-------------+-------------+-------------|
| total           |      146.11 |       39.92 |        3.7x |
```

<https://elpa.gnu.org/packages/elisp-benchmarks.html>

## libgccjit take aways

Works for us!

- Compile time is okay
- Leaks memory
- How to expose easily more accessors? LTO?
- Distros may fix their packages

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

## Project Status

- Exists and it is usable

```
emacs.git feature/native-comp
./configure --with-nativecomp
M-x report-emacs-bug
```

- Focusing on integration and consolidating
- Maybe in Emacs 28?

<http://akrl.sdf.org/gccemacs.html>

<emacs-devel@gnu.org>

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

## Project Status

- Exists and it is usable

```
emacs.git feature/native-comp

./configure --with-nativecomp

M-x report-emacs-bug
```

- Focusing on integration and consolidating
- Maybe in Emacs 28?

<http://akrl.sdf.org/gccemacs.html>

<emacs-devel@gnu.org>