Common Lisp ecosystem and the software distribution model

Daniel Kochmański

TurtleWare

July 3, 2016

- 1936 – Alozno Church invents Lambda Calculus
- 1960 – John McCarthy presents paper about LISP
- 1973 – MIT Lisp Machine Project
- 1984 – AI winter (and the unfortunate marriage with Lisp)
- 1994 – Various dialects unification with Common Lisp
- 2000 – Renaissance of the community

Figure: "John McCarthy presents Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I" – Painting by Ferdinand Bol, 1662

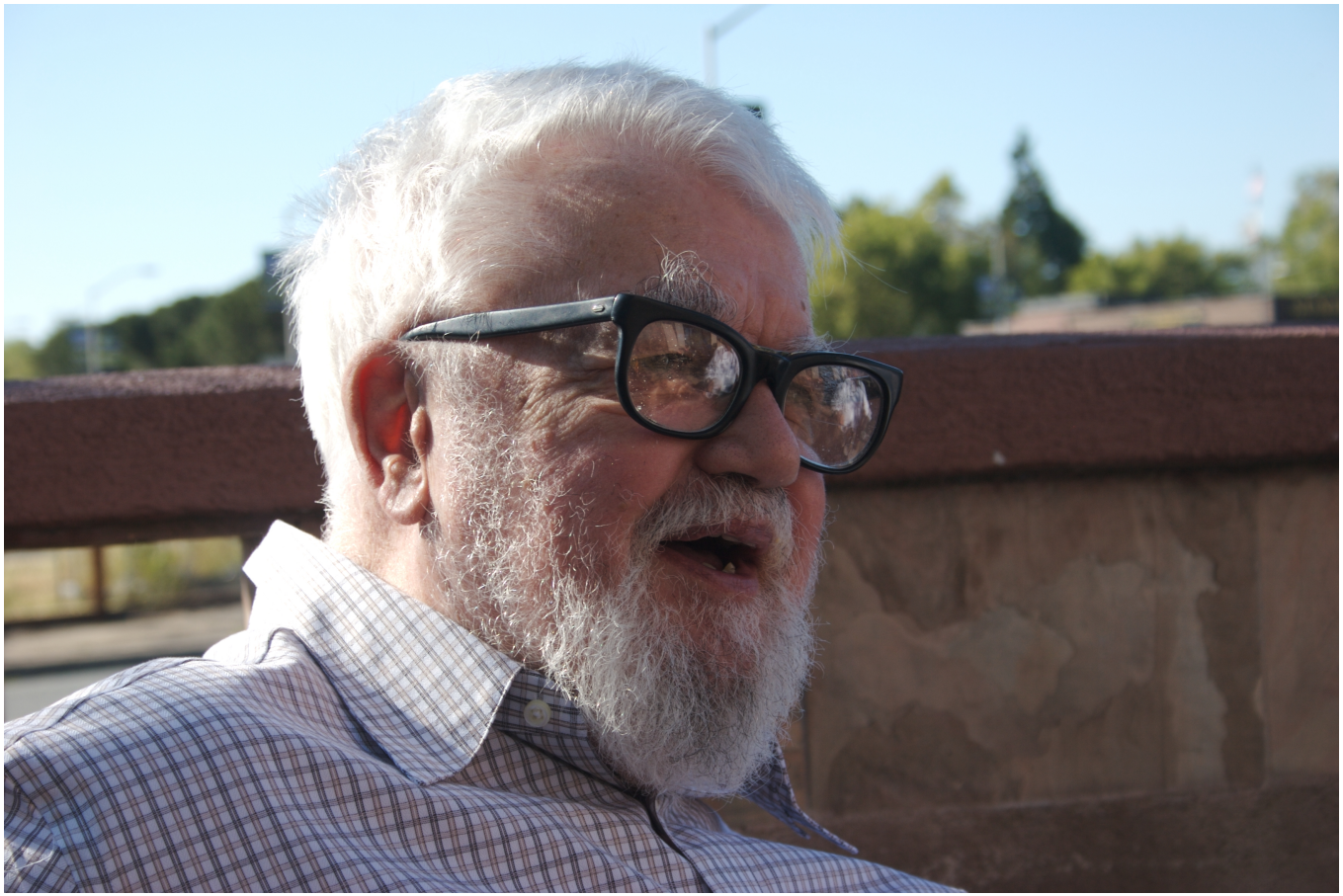

Figure: John McCarthy (1927-2011)

Figure: A Knight machine preserved in the MIT Museum

	1955	1960	1965	1970	1975	1980	1985	1990	1995	2000	2005	2010	2015
Lisp 1.5	Lisp 1.5												
Maclisp			Maclisp										
Interlisp				Interlisp									
ZetaLisp					Lisp Machine Lisp								
Scheme					Scheme								
NIL					NIL								
Common Lisp						Common Lisp							
T							T						
AutoLISP							AutoLISP						
ISLISP							ISLISP						
EuLisp									EuLisp				
Racket									Racket				
Arc										Arc			
Clojure											Clojure		
LFE											LFE		
Hy												Hy	

Figure: Lisp dialects timeline

- Homoiconicity – code is a data, everything is an expression
- Macros – growing the language
- Higher-order and anonymous functions
- Flexible type system
- Garbage collection
- Read-Eval-Print-Loop – interactive programming
- The whole language is always available
- CLOS – generic function-style OOP

Common Lisp function definition

```
(defun compose (function &rest more-functions)
  (declare (optimize (speed 3) (safety 1) (debug 1)))
  (reduce
   (lambda (f g)
     (let ((f (ensure-function f))
           (g (ensure-function g)))
       (lambda (&rest arguments)
         (declare (dynamic-extent arguments))
         (funcall f (apply g arguments)))))
   more-functions
   :initial-value function))
```

Common Lisp macro definition

```
(defmacro while (test &body body)
  (let ((ret (gensym)))
    `(block nil
       (do ((,ret nil (progn ,@body)))
           ((not ,test) ,ret)))))
```

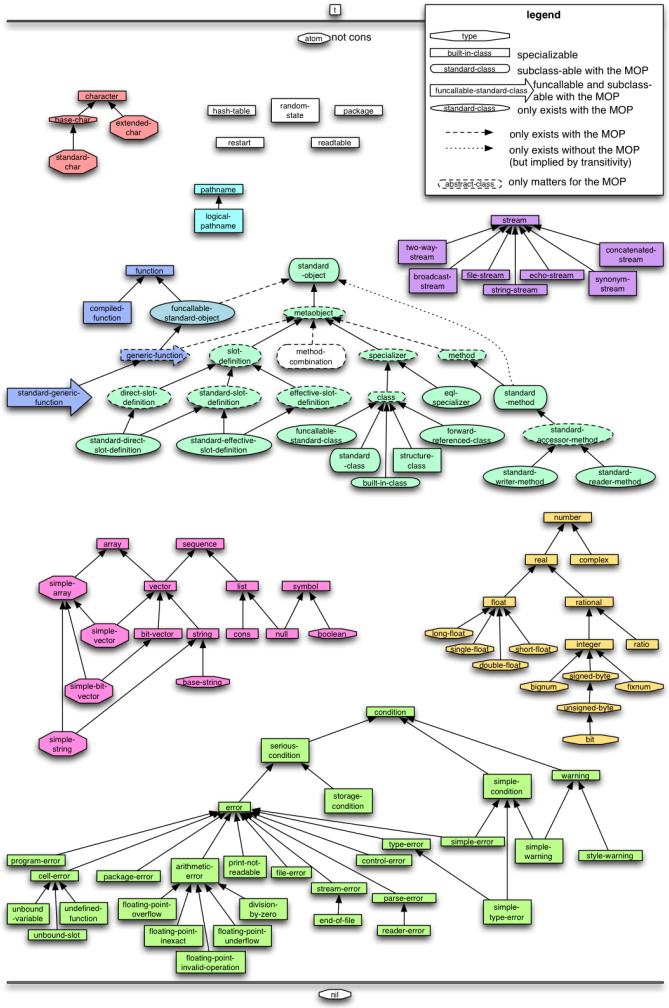

Figure: Common Lisp types hierarchy

- Wide range of implementations
- Active FOSS community
- Growing ecosystem

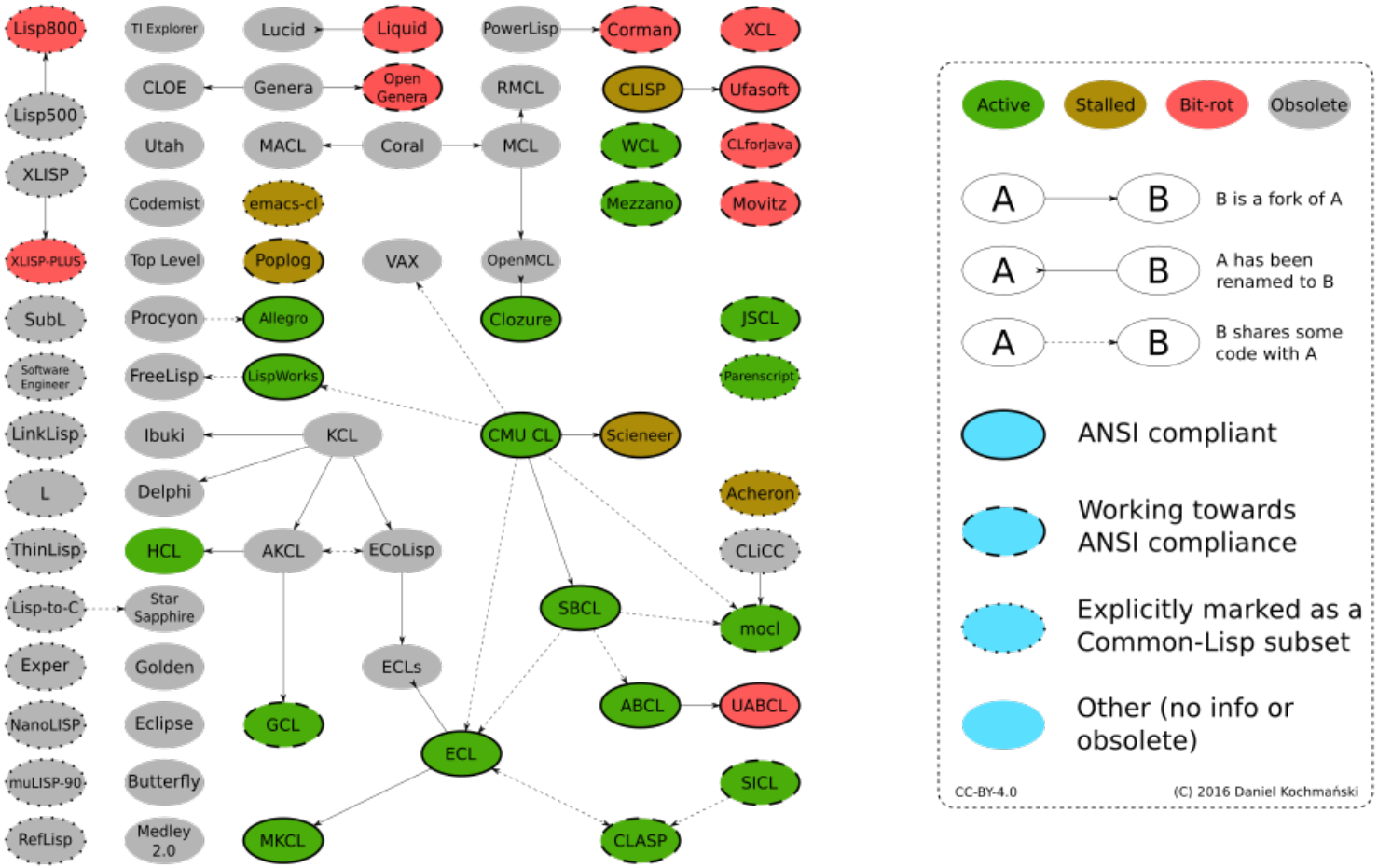

Figure: Common Lisp implementations graph

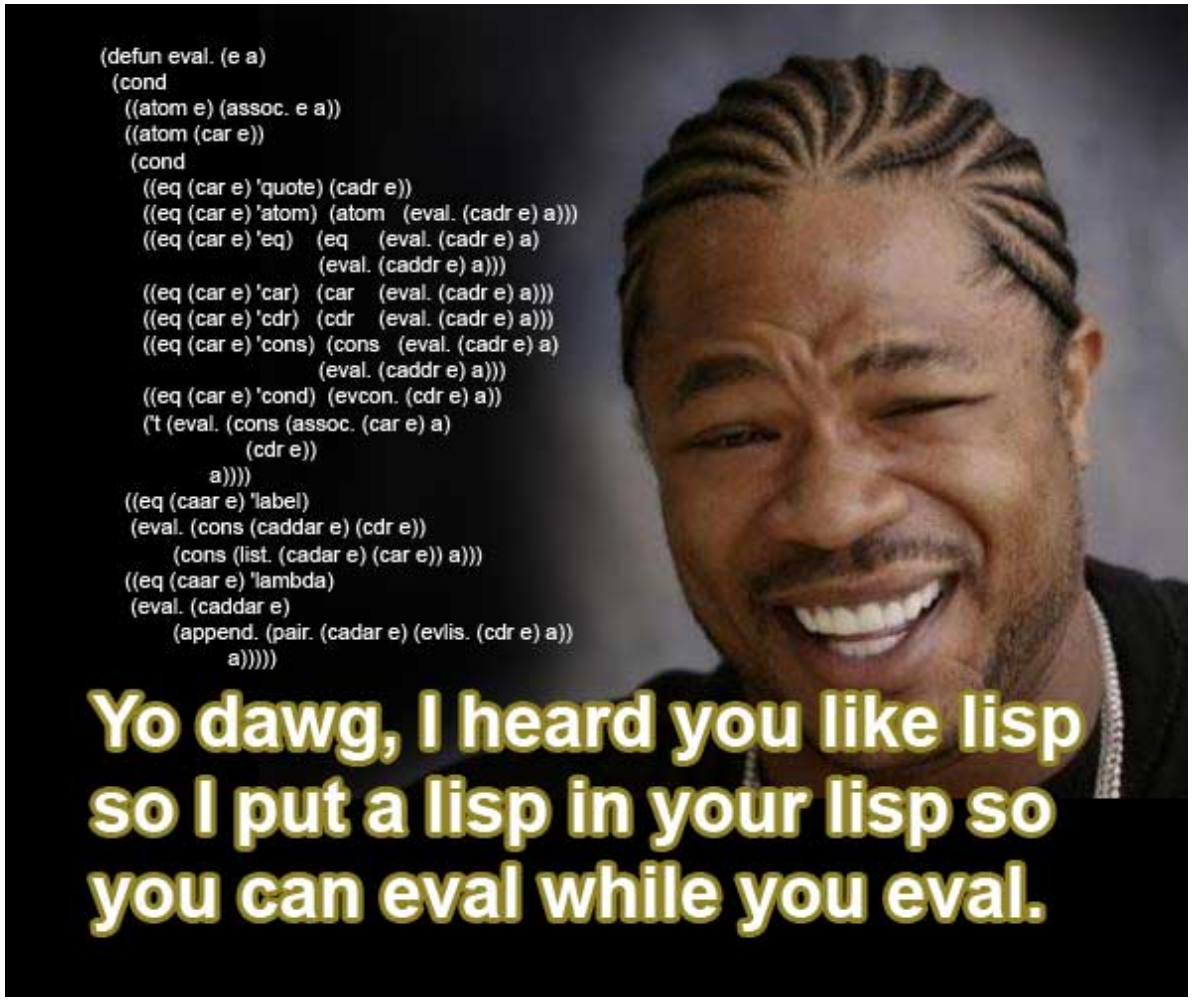

Figure: Typical Lisp programmer

Incremental compilation

```
CL-USER> (defun yyy () (xxx))
YYY
CL-USER> (defun xxx () "Hello world")
XXX
CL-USER> (yyy)
"Hello world"
CL-USER> (defun xxx () "Bye world")
xxx
CL-USER> (yyy)
"Bye world"
```

FASt Load

```
CL-USER> (compile-file "xxx.lisp")
"/home/jack/xxx.fas"

CL-USER> (load "xxx")
T

CL-USER> (xxx)
"Hello world"
```

save-lisp-and-die and build-system

```
CL-USER> (sb-ext:save-lisp-and-die "xxx")

[jack@pandora ~]$ ./xxx
"Hello world"

CL-USER> (compiler::builder
           :program "xxx"
           :lisp-files `("file-1.lisp"
                         "file-2.lisp")
           :main-name "main")

[jack@pandora ~]$ ./xxx
"Hello world"
```

Deployment

The following facilities are wrappers around save-lisp-and-die or build-system (via ASDF which is covered later):

- cl-launch
- clbuild
- clon
- roswell
- uiop

Manual system definition example

```
(defparameter *source-files*
  '("packages"
    "utilities"
    "classes"
    "application"))

(mapcar #'(lambda (f)
            (format t "Loading file ~A~%" f)
            (load f))
        *source-files*)
```

- defsystem, mk-defsystem and sbt-defsystem
- ASDF – Another System Definition Facility
- ISDF – Inferred System Description Facility

```
(defsystem #:metering
  :name "Metering" :version "3.2"
  :description "Portable Code Profiling Tool"
  :author "Mark Kantrowitz <mkant@cs.cmu.edu>"
  :maintainer "Daniel Kochmański <daniel@turtleware.eu>"
  :components ((:cl-source-file.cl "metering"))
  :in-order-to ((test-op (test-op #:metering/test))))

(defsystem #:metering/test
  :depends-on (#:metering #:fiveam)
  :components ((:file "metering-test"))
  :perform (test-op (o s)
             (funcall (intern (string '#:run!)
                              :metering/test)
                      :metering-suite)))
```

Dependency graph example

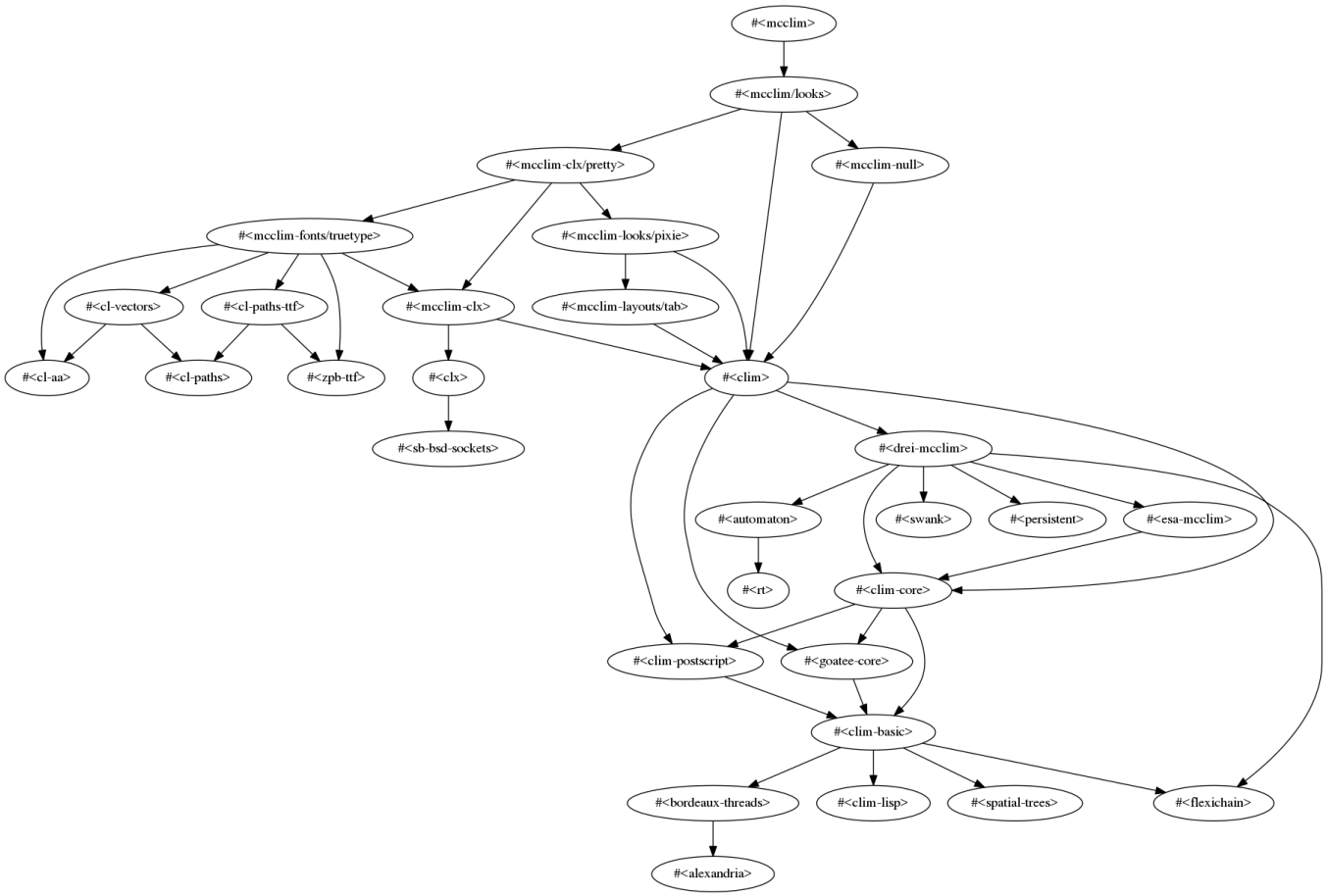

Figure: McCLIM dependencies

State of the art

A few software distribution solutions out there:

- common-lisp-controller
- asdf-install
- Quicklisp

- NPM (JavaScript)
- RubyGems (Ruby)
- CPAN (Perl)

- aptitude
- guix
- pkgsrc
- portage
- NiX

Usage example

```
CL-USER> (ql:quickload 'clim-examples)
T

CL-USER> (clim-demo:demodemo)
; magic happens
```

Pros and cons

- Easy to use
- Well maintained
- Allows custom repositories
- Reliable
- Integrated with the language

- Poor documentation
- Single trust authority (not safe!)

Demo

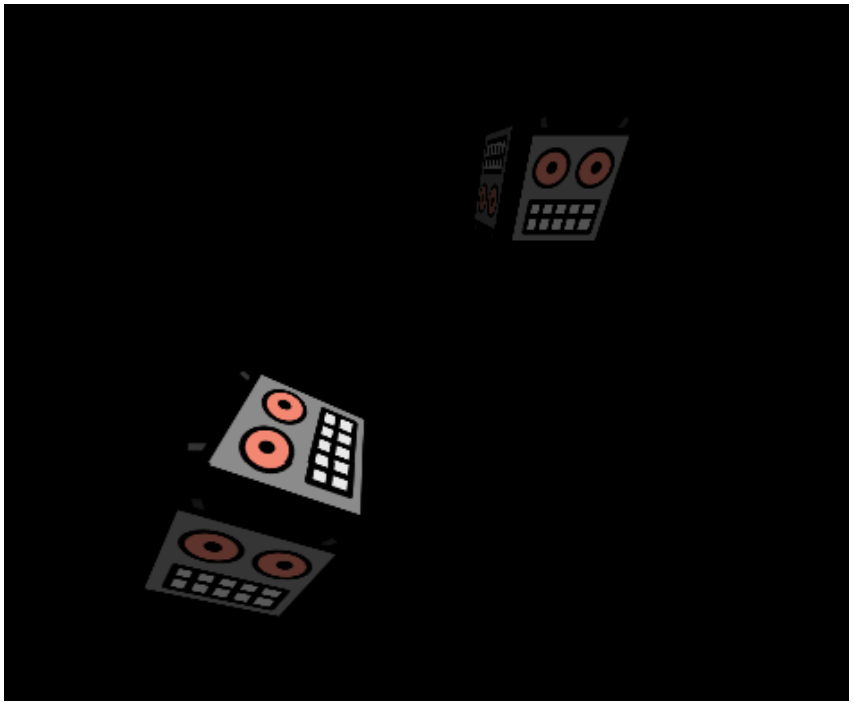

Figure: CLinch Demo

Common Lisp introduction

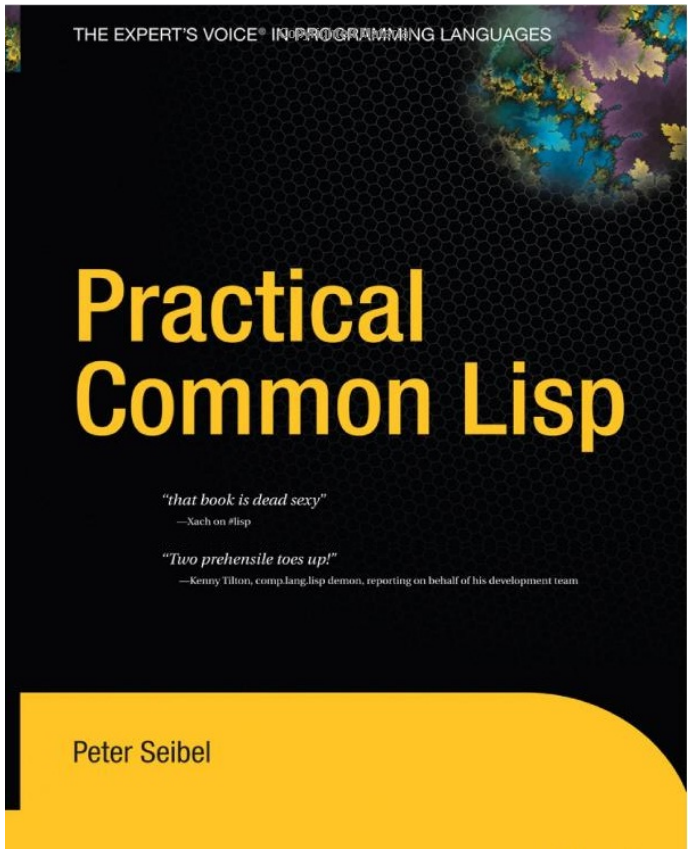

Figure: Practical Common Lisp – Peter Seibel

Summary

- Lisp has quite a history
- Its features are slowly adapted in other languages
- Common Lisp has various implementations
- Distributing binaries isn't a viable option for the CL *developers* (many binary-incompatible implementations)
- Quicklisp is in a similar spirit as the NPM and the RubyGems

Literature

- Lisp: Good News, Bad News, How to Win Big – Richard P. Gabriel [https://www.dreamsongs.com/WIB.html]
- Revenge of the Nerds – Paul Graham [http://paulgraham.com/icad.html]
- Beating the Averages – Paul Graham [http://paulgraham.com/avg.html]
- Practical Common Lisp – Peter Seibel [http://www.gigamonkeys.com/book/]

Attributions

- "Lisp" painting – http://classicprogrammerpaintings.com/post/142817850864/john-mccarthy-presents-recursive-functions-of
- John McCarthy photo (*CC BY-SA 2.0*) by *null0* at http://www.flickr.com/photos/null0/272015955/
- Lisp Machine photo (*CC BY-SA 3.0*) – no machine-readable author provided. Jszigetvari assumed (based on copyright claims)
- Lisp dialects (*CC BY-SA 3.0*) – https://en.wikipedia.org/wiki/Lisp

Attributions

- Common Lisp types hierarchy (CC BY-NC-SA 3.0) by Greg Pfeil, http://sellout.github.io/2012/03/03/common-lisp-type-hierarchy/
- Common Lisp implementations graph (*CC-BY-4.0*) by Daniel Kochmański, https://common-lisp.net/project/ecl/posts/ECL-Quarterly-Volume-IV.html
- Typical Lisp programmer photo – http://jonex.info/dump/yolisp.jpg (based on http://community.schemewiki.org/?scheme-fortune-cookies)
- "Practical Common Lisp" cover – http://www.gigamonkeys.com/book/

Contact

Daniel Kochmański [daniel@turtleware.eu] [jackdaniel@freenode]

About me

I build device prototypes and do FOSS consultancy in my own company TurtleWare. I specialize in the embedded systems, Linux kernel and userspace development, C/C++ and Common Lisp programming and compiler design.

This presentation is available at
http://turtleware.eu/static/talks/pkgsrcCon-2016-lisp.pdf