LISP Introduction and European Lisp Symposium 2016

Alexey Veretennikov
alexey@veroveli.com

Veroveli AB

June 8, 2016

What is LISP?

- Introduced in 1958 by John McCarthy for symbolic data manipulation
- After FORTRAN is the oldest language still in use
- Design goals:
  - Dynamic - type follows the value, not variable
  - Functional - functions as a first-class entities (as a data)
  - Allows to manipulate symbolic data
  - Garbage-collected
  - LISP means “LISt Processing”

Usage

LISP major domain fields:

- (original) AI systems
- Prototyping
- Science
- Complex systems
- (web) Backends
- General-purpose language, so anything!

Dialects

LISP is a family of languages nowadays. The major general-purpose dialects in this family are:

- **Common Lisp** (ANSI standard 1994). Actual compilers/interpreters:
  - SBCL (opensource): `http://www.sbcl.org/`
  - Clozure CL (opensource): `http://ccl.clozure.com/`
  - ABCL (opensource, JVM): `http://abcl.org/`
  - LispWorks (commercial): `http://www.lispworks.com/`
  - Allegro CL (commercial): `http://franz.com/products/allegro-common-lisp/`
- **Scheme** (IEEE standard, with revisions latest R6RS 2007)
  - Major spin-off dialect: **Racket** `http://racket-lang.org/`
- **Clojure** (JVM-based) `http://clojure.org/`
- **GNU Emacs Lisp** :) `https://www.gnu.org/software/emacs/`

Hype and Reality

The language surrounded by a big hype:

- Tend to be considered as a silver bullet in 1980s
- Language for a geeks/extremely smart guys "elite hackers" - this largely the result of the popular article *Beating the averages* by Paul Graham, the prominent Lisp hacker `http://www.paulgraham.com/avg.html`
- Tons of ')'
- Is slow

However the reality is

- Interest generally faded after "AI winter" of 1990s
- Huge legacy and not enough tooling
- Bad state of libraries - a lot of abandoned

Modern usage examples

Commercial, the representatives I've met on a conference:

- Google ITA: Intelligent flight/fare solution `https://www.itasoftware.com/`
- Grammarly: Online grammar and spelling checker `https://www.grammarly.com/`
- RavenPack: Automatic filtering and conversion of the news to financial applications `http://www.ravenpack.com/`

Educational:

- *Structure and Interpretation of Computer Programs* `https://www.mitpress.mit.edu/sicp/full-text/book/book.html`, classical MIT course
- Computer Science courses around Europe teaching in Racket

Also used in Science labs around the globe for basically anything: from big-data to chemistry

Properties of the language

- Everything is either a **list** or **atom**
- Uniform syntax
- The data representation is the same as code representation:
  - Therefore the data could be a code and a code $\rightarrow$ data
  - Code could be generated and manipulated in run-time
- Image-based
- REPL (Read Eval Print Loop) $\rightarrow$ **interactive development**
- Multi-paradigm: Functional, Procedural and Object-Oriented
- Macros is a part of language to build the DSLs on top of the language

Syntax example

- Core of the language: S-expressions (Sexpr):

```
(item1 item2 item3)
```

- Prefix notation only:

```
(+ 1 2 3)
```

- Nothing more!

```
(setf x 1)
;; (if condition then-stmt [else-stmt])
(if (> x 10)
     (print x)
   (print ``less than 10''))
```

Functions

Functions

```
(defun add (x y)
  (+ x y))
;; call it
(add 1 2)
```

Anonymous functions

```
((lambda (x y) (+ x y)) 1 2)
```

Functional programming

Function as an argument

```
(defun two-args (fun x y)
   (funcall fun x y))
```

Called like this:

```
(two-args #'+ 1 2)
(two-args #'add 1 2)
(two-args (lambda (x y) (+ x y)) 1 2)
```

Function as a result:

```
(defun make-adder (x) (lambda (y) (+ x y)))
;; call it:
(funcall (make-adder 1) 2)
=> 3
```

Lexical Closures 1

Lexical environment:

```
(let ((x 1)
      (y 2))
  (+ x y))
=> 3
```

Lexical closure:

```
(let ((counter 0))
   (defun f (x)
      (setf counter (+ 1 counter))
      (+ x counter)))
(f 1)
=> 2
(f 1)
=> 3
(f 1)
=> 4
```

Lexical Closures 2

```
(defun make-counter ()
  (let ((counter 0))
    (lambda (x)
      (setf counter (+ 1 counter))
      (+ x counter))))
;; use it
(setf fun1 (make-counter))
(setf fun2 (make-counter))
(funcall fun1 1)
=> 2
(funcall fun1 1)
=> 3
(funcall fun2 1)
=> 2
```

Lists

```
(setf a (list 1 "hello" 'world))
=> (1 "hello" WORLD)
(car a)
=> 1
(cdr a)
=> (2 3)
(cons 1 '(2 3))
=> (1 2 3)
(list 'one a 'three)
=> (ONE (1 "hello" WORLD) THREE)
(reverse '(1 2 3) ;; or (reverse (quote (1 2 3)))
=> (3 2 1)
(find-if (lambda (x) (> x 3)) '(1 3 4 -2))
=> 4
(append '(if (> 1 2)) '(1 2))
=> (IF (> 1 2) 1 2)
```

Lexical Closures 3

```
(defun make-point ()
  (let ((x 0) (y 0))
    (list
     (lambda () x)
     (lambda (x1) (setf x x1))
     (lambda () y)
     (lambda (y1) (setf y y1)))))
;; try to use
(setf a (make-point))
(funcall (first a))
=> 0
(funcall (second a) 1)
(funcall (first a))
=> 1
```

Lexical Closures 4

```
(defun get-x (point)
  (funcall (first point)))
(defun get-y (point)
  (funcall (third point)))
(defun set-x (point x)
  (funcall (second point) x))
(defun set-y (point y)
  (funcall (fourth point) y))
;; use them!
(get-x a)
=> 1
(set-x a 10)
(get-x a)
=> 10
(get-x (make-point))
=> 0
```

Metaprogramming: EVAL

```
(eval '(+ 1 2))
=> 3
(if (> 2 1) "bigger" "not bigger")
=> "bigger"
(reverse '("not bigger" "bigger" (> 2 1) if)))
=> (IF (> 2 1) "bigger" "not bigger")
(eval (reverse '("not bigger" "bigger" (> 2 1) if)))
=> "bigger"
```

Metaprogramming: Macros

- Example:

```
(defmacro aif (stmt then else)
  (list 'if stmt then else))
;; examine it
(macroexpand '(aif (> 1 2) 1 2))
=> (IF (> 1 2) 1 2)
;; use it
(aif (> 1 2) 1 2)
=> 2
```

- Use the full language power to produce compile-time expansion (unlike C macros which are just text preprocessor)
- Allows to build language extensions: new operators etc
- Standard is not moving, but language evolves via libraries and macros

European Lisp Symposium 9-10 May 2016

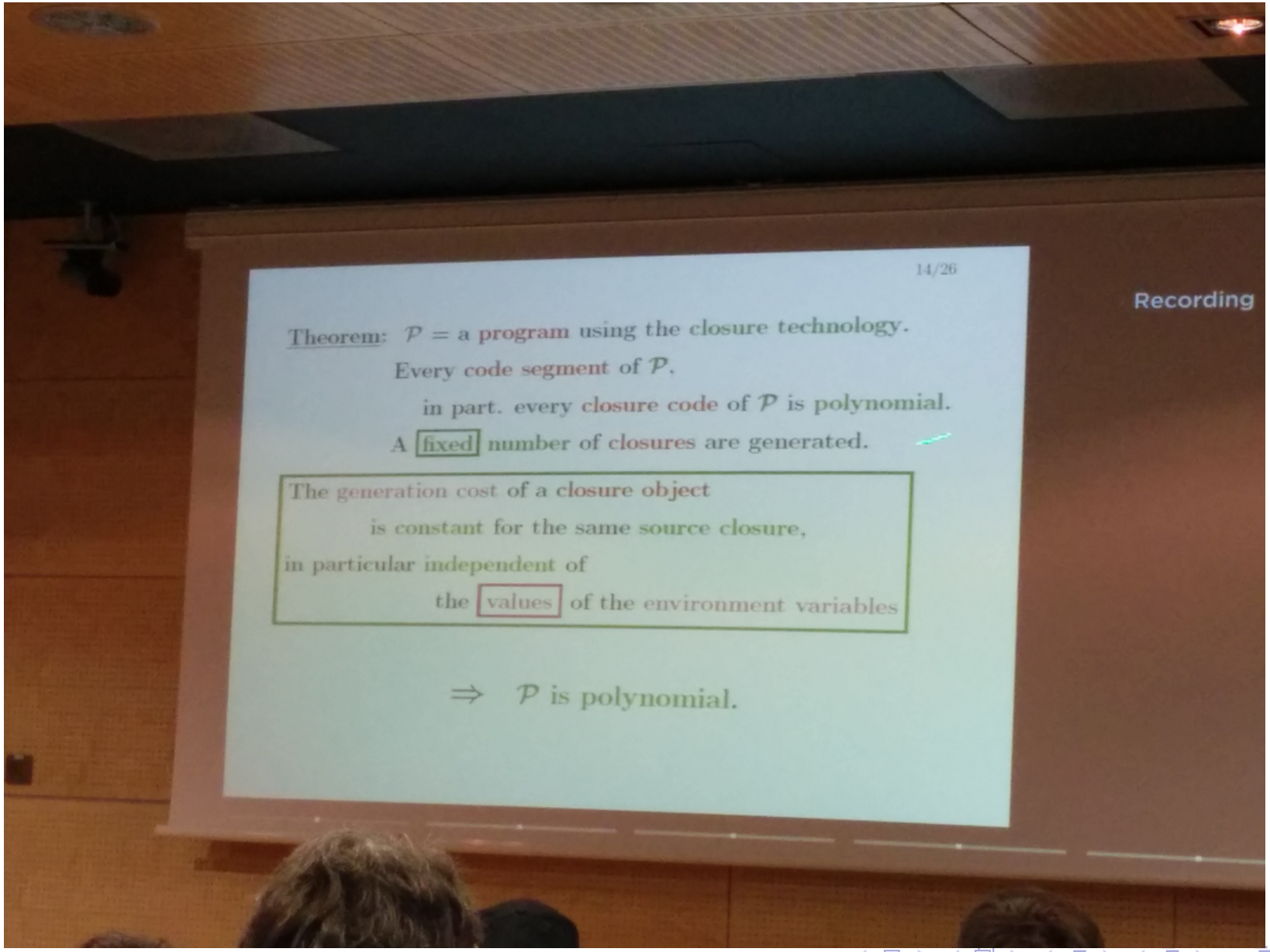
- The 9th European Lisp Symposium
- Held in Krakow, Poland, AGH University of Science and Technology, Department of Computer Science

ELS2016 most interesting topics (to me)

- Keynote I: Lexical Closures and Complexity
- A High-Performance Image Processing DSL for Heterogeneous Architectures
- Fast Interactive Functional Computer Vision with Racket
- CANDO: A Compiled Programming Language for Computer-Aided Nanomaterial Design and Optimization Based on Clasp Common Lisp

ELS2016: Keynote I: Lexical Closures and Complexity

By *Francis Sergeraert, Institut Fourier, Grenoble, France*



ELS2016: A High-Performance Image Processing DSL for Heterogeneous Architectures: 1

By *Kai Selgrad, Alexander Lier, Jan Drntlein, Oliver Reiche Marc Stamminger, Computer Graphics Group, Hardware/Software Co-Design Friedrich-Alexander University Erlangen-Nuremberg, Germany*

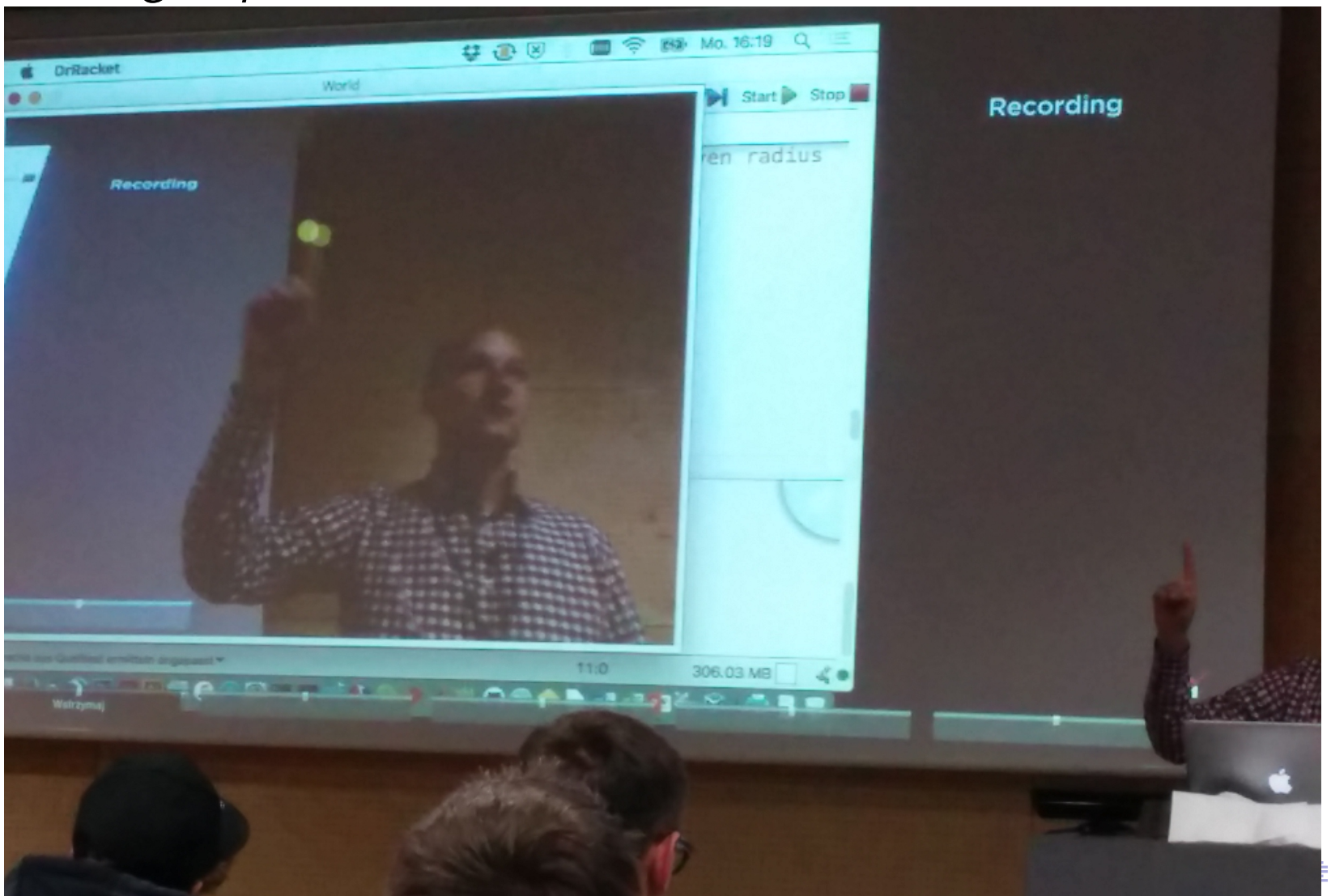
ELS2016: A High-Performance Image Processing DSL for Heterogeneous Architectures: 2

Example:

```
(defilter filter (data mask (w h) (filter-w filter-h)) (
(decl ((float accum 0.0f))
  (loop2d (dx dy filter-w filter-h)
    (set accum (* (mask (cell
      dx dy)
         (+ x (- (/ filter-w 2)) dx)
         (+ y (- (/ filter-h 2)) dy)))))
  (set (cell x y) accum))))
```

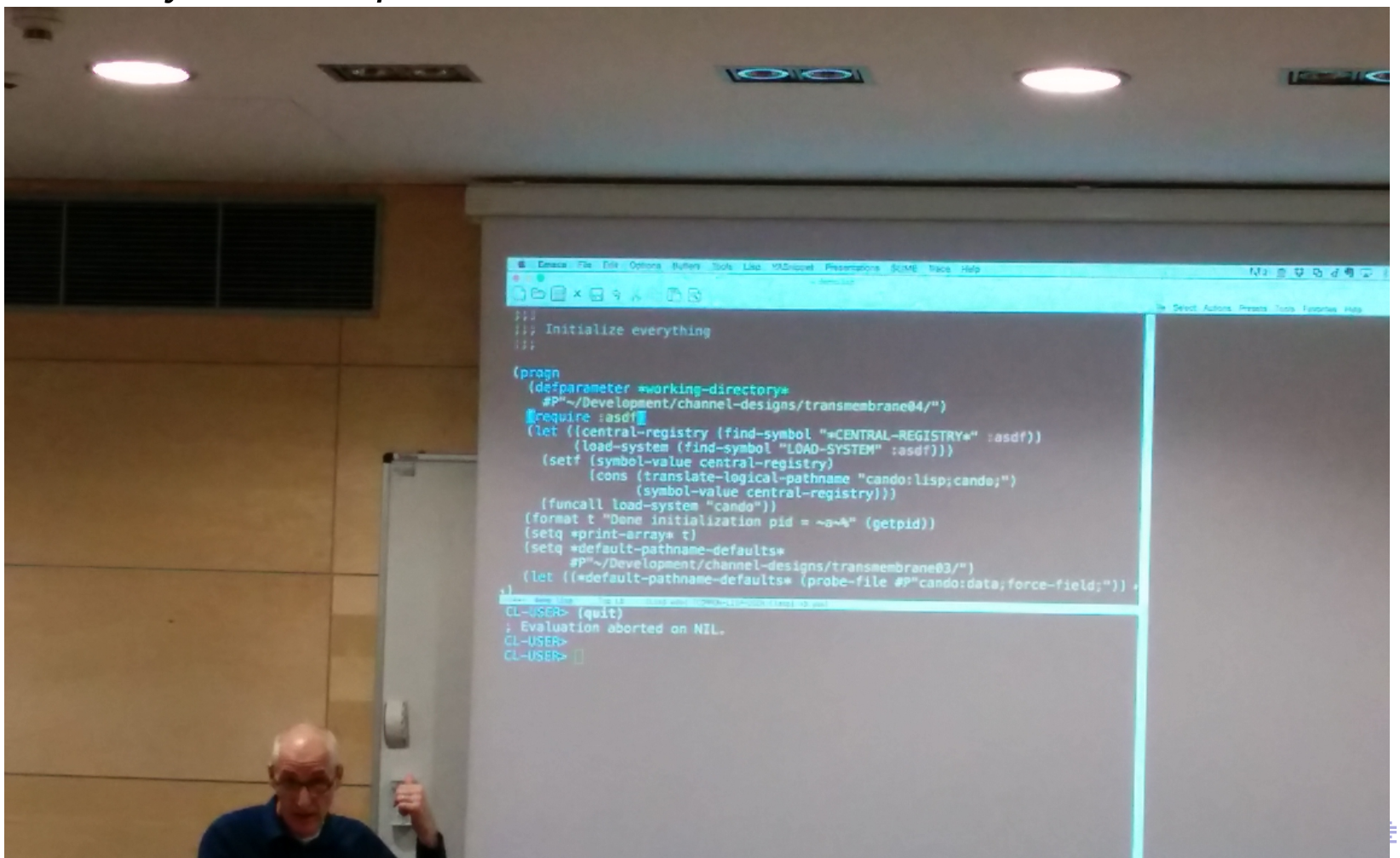
ELS2016: Fast Interactive Functional Computer Vision with Racket

By *Benjamin Seppke, Leonie Dreschler-Fischer, University of Hamburg Dept. Informatics*



ELS2016: CANDO: A Compiled Programming Language for Computer-Aided Nanomaterial Design and Optimization Based on Clasp Common Lisp

By *Christian E. Schafmeister, Chemistry Department, Temple University Philadelphia*



Fun part: Conference Dinner

The end

Questions?