Lightning Talks II

ELS 2016

- **01 ILLITHID *- Mark Evenson***
- 02 Sugaring Lisp for the 21st Century - *Vsevolod Dyomkin*
- 03 Common Lisp UltraSpec - *Michał Herda*
- 04 PRIMSMA - *Philip Gagner*
- 05 Managing Packages - *James Anderson*
- 06 CEPL Reaches Beta - *Chris Bagley*
- 07 QueryFS: a Solution Looking for More Problems - *Michael Raskin*
- 08 LISP/c - *Michał Herda (on behalf of Jonathan Baca)*
- 09 Electricity is Orange - *Devon McCullough*

MARK <EVENSON.NOT.ORG@GMAIL.COM>
ELS 2016 / KRAKOW, POLAND

ILLITHID: A REPORT ON USING GOING INTO PRODUCTION WITH ABCL

ILLITHID

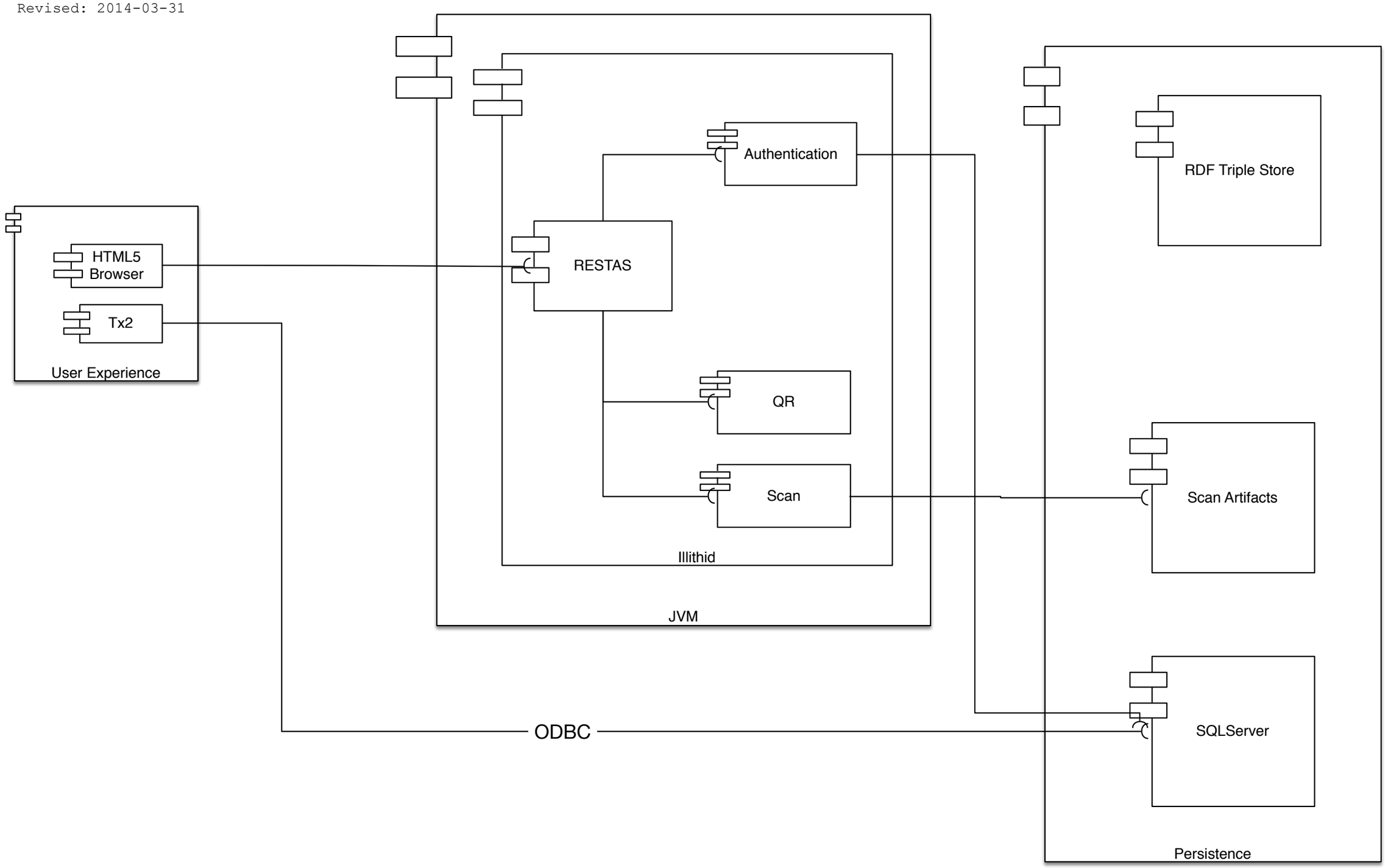
- (“Picasso”) Electronic Health Record system
  ~20 year old codebase University at Buffalo School of Dental Medicine
  used by ~1000 Students ~600 Dentists
  In-house custom software based on PowerBuilder (4GL with SQL persistence)
  Application has both record keeping/pedagogical (grading)
- (“Pablo”) Incremental “Brownfield” project begun in Fall 2013; first production in May 2014
  - Move to HTML5 client (while obeying HIPAA regulation)
  - Move to use of OWL2 Ontologies for data description
    - 1) To facilitate research across anonymized data sets
    - 2) To enable interoperability with other medical data systems

ILLITHID

- Illithid: “REST Broker” built on ABCL
  - QuickLisp: Hunchentoot, RESTAS, CL-WHO, ParenScript, LParallel, CXML, XPath
    Total of around ~60 ASDF systems
  - Java Libraries: Render to PDF, SQL connectivity
  - LSW2 (Alan Rutenberg https://github.com/alanruttenberg/lsw2)
    OWL2-based Reasoner; data persisted as RDF
  - Ontotext GraphDB (OWLIM)
  - Microsoft SQL Server

ILLITHID

UBSDM “Pablo” Picasso
Mark Evenson
Created: 2014-03-31
Revised: 2014-03-31



ILLITHID

- Problem
  Need a simplified deployment strategy for Microsoft Windows because system operators “Don’t want to know about Lisp” (i.e. no management via REPL)
- Solution
  Use Java Servlet Container (Tomcat) instances that manages the application lifecycle (start, stop, logging, deployment descriptor for database location)
  - Need to provide a single Java deployment artifact (“Web Application Archive”) to system operators
- ABCL-SERVLET <https://bitbucket.org/easye/abcl-servlet>
  Scaffolding for a Java Servlet that boots ABCL, and executes configurable Lisp Code

Creating Java Deployment Artifact 1/2

- All software components encapsulated by ASDF
- (ABCL-ASDF) ABCL extension to ASDF uses Maven to describe Java software library dependent which downloads the Java dependencies upon first run

```
(:module maven
            :components
            ((:mvn "org.apache.xmlgraphics/fop/1.0")
             (:mvn "org.apache.pdfbox/pdfbox/1.8.4")
```

Creating Java Deployment Artifact 2/2

- (ASDF-JAR) ABCL extension to locate ASDF systems within the deployment artifact
  - Simple packaging strategy:
    Assuming that all ASDF systems have a “root” description file, locate the ASDF root on the filesystem, recursively copy all components (including Quicklisp)
- Result: 108 Mib “illithid.war” deployment artifact
- Bonus: able to connect to production application via Swank

- 01 ILLITHID - *Mark Evenson*
- **02 Sugaring Lisp for the 21st Century - *Vsevolod Dyomkin***
- 03 Common Lisp UltraSpec - *Michał Herda*
- 04 PRIMSMA - *Philip Gagner*
- 05 Managing Packages - *James Anderson*
- 06 CEPL Reaches Beta - *Chris Bagley*
- 07 QueryFS: a Solution Looking for More Problems - *Michael Raskin*
- 08 LISP/c - *Michał Herda (on behalf of Jonathan Baca)*
- 09 Electricity is Orange - *Devon McCullough*

Sugaring Lisp for the 21st Century

Vsevolod Dyomkin
@vseloved

European Lisp Symposium
2016-05-09

Common Complaints about Common Lisp

(from the POV of outsiders)

~~* No libraries~~
~~* No threads, sockets, whatever in the standard~~
* “Historical cruft” (poor names, verbosity, inconsistency, lack of modularity, not generic enough etc.)

“Modernization” Efforts

* CDR https://common-lisp.net/project/cdr/
Common Lisp Document Repository

* cltl3 http://ilc2009.scheming.org/node/48
“Codifying Modern Common Lisp”

* cl21 https://github.com/cl21/cl21
“Common Lisp in the 21st Century”

Beyond Utilities

Default solution — utilities (http://cliki.net/utilities - 36 entries of which at least 10+ are general-purpose ones).

Most don't go far enough — added value is only convenience, not language evolution and growth

RUTILS

est. 2009

14 core + 7 contrib packages
~ 250 exported symbols
(cf. ~1000 symbols in package CL)

* backward-compatible
* practical
* modular
* pro-choice

https://github.com/vseloved/rutils

Example Code

```
(defmethod select-transition ((parser amrparser) graph)
  (with (((stack buffer) @ parser))
        (s0 (? stack 0))
        (b0 (? buffer 0)))
    (flat-map (lambda (k)
                (unless (or (in# (pair k b0) graph)
                            (in# (pair b0 k) graph))
                  (let ((fs (append common-fs
                                    (extract-fs parser
                                     nil s0 b0))))
                    (mapcar ^(make-trans :fn % :fs fs)
                            '(amr:reattach
                              amr:reenter)))))
              (set-difference @parser.tokens
                              (list nil s0 b0))))))
```

```
CL-USER> (select-transition (make 'parser) #h(equal))
```

How You Can Benefit

* Use it

```
;; conventional way
(ql:quickload :rutils)
(use-package :rutils)
(named-readtables:in-readtable rutils-readtable)

;; radical way
(ql:quickload :rutilsx)
(use-package :rutilsx)
(named-readtables:in-readtable rutilsx-readtable)
```

How You Can Benefit

* Use parts of it
  ```
  (use-package :rutilsx.threading)
  ```

* Borrow ideas ...like cl21 did
  usually called cross-pollination

Open Issues

How to gracefully handle:

* map/mapcar
* slot-value names across packages in @obj.slot

- 01 ILLITHID - *Mark Evenson*
- 02 Sugaring Lisp for the 21st Century - *Vsevolod Dyomkin*
- **03 Common Lisp UltraSpec - *Michał Herda***
- 04 PRIMSMA - *Philip Gagner*
- 05 Managing Packages - *James Anderson*
- 06 CEPL Reaches Beta - *Chris Bagley*
- 07 QueryFS: a Solution Looking for More Problems - *Michael Raskin*
- 08 LISP/c - *Michał Herda (on behalf of Jonathan Baca)*
- 09 Electricity is Orange - *Devon McCullough*

Yet Another Rant About The State Of Common Lisp Documentation

Michał „phoe” Herda

Common Lisp HyperSpec™

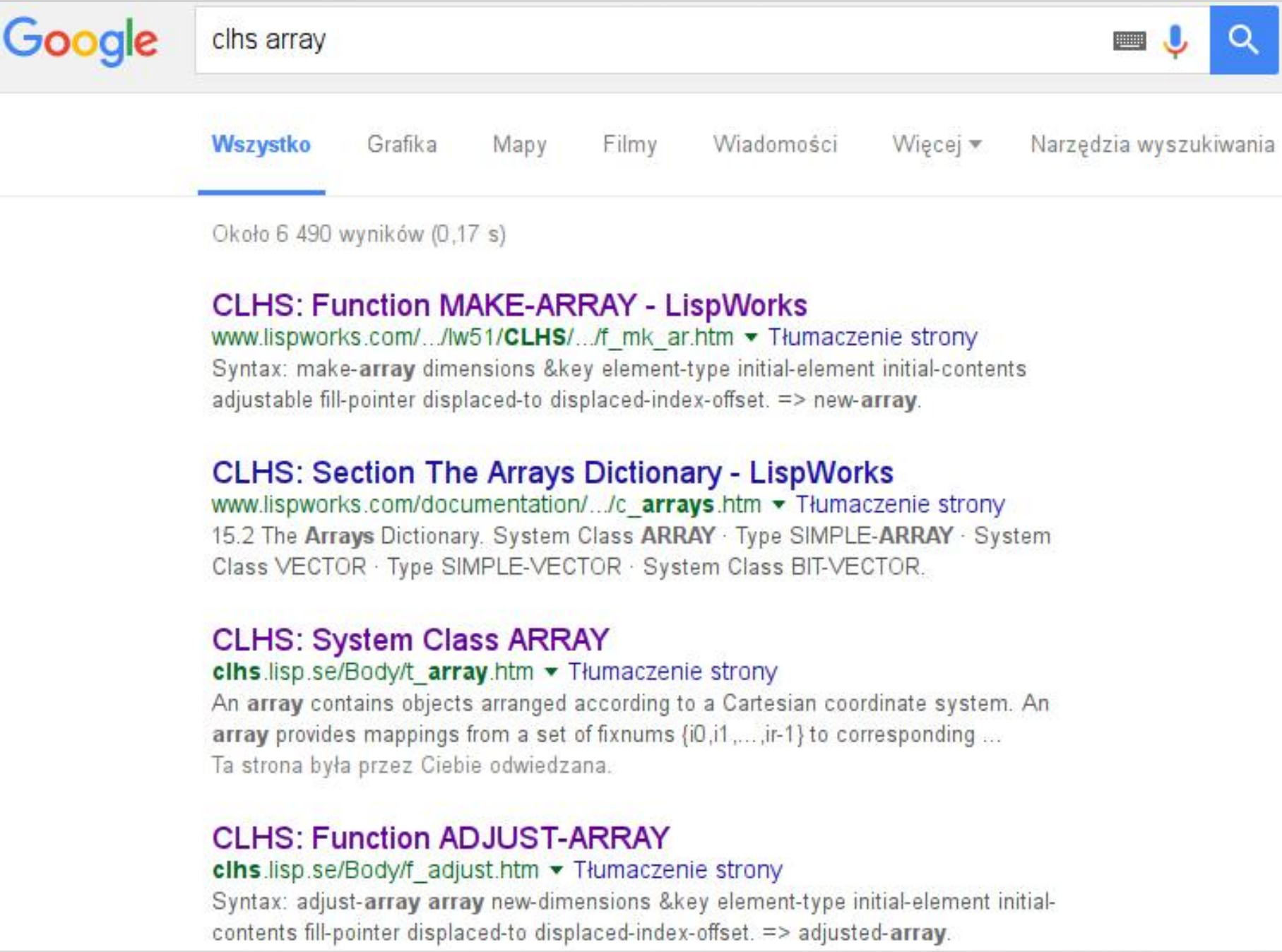
The very definition of class.

Welcome to the *Common Lisp HyperSpec*.
I hope it serves your need.

--*Kent Pitman*, X3J13 Project Editor

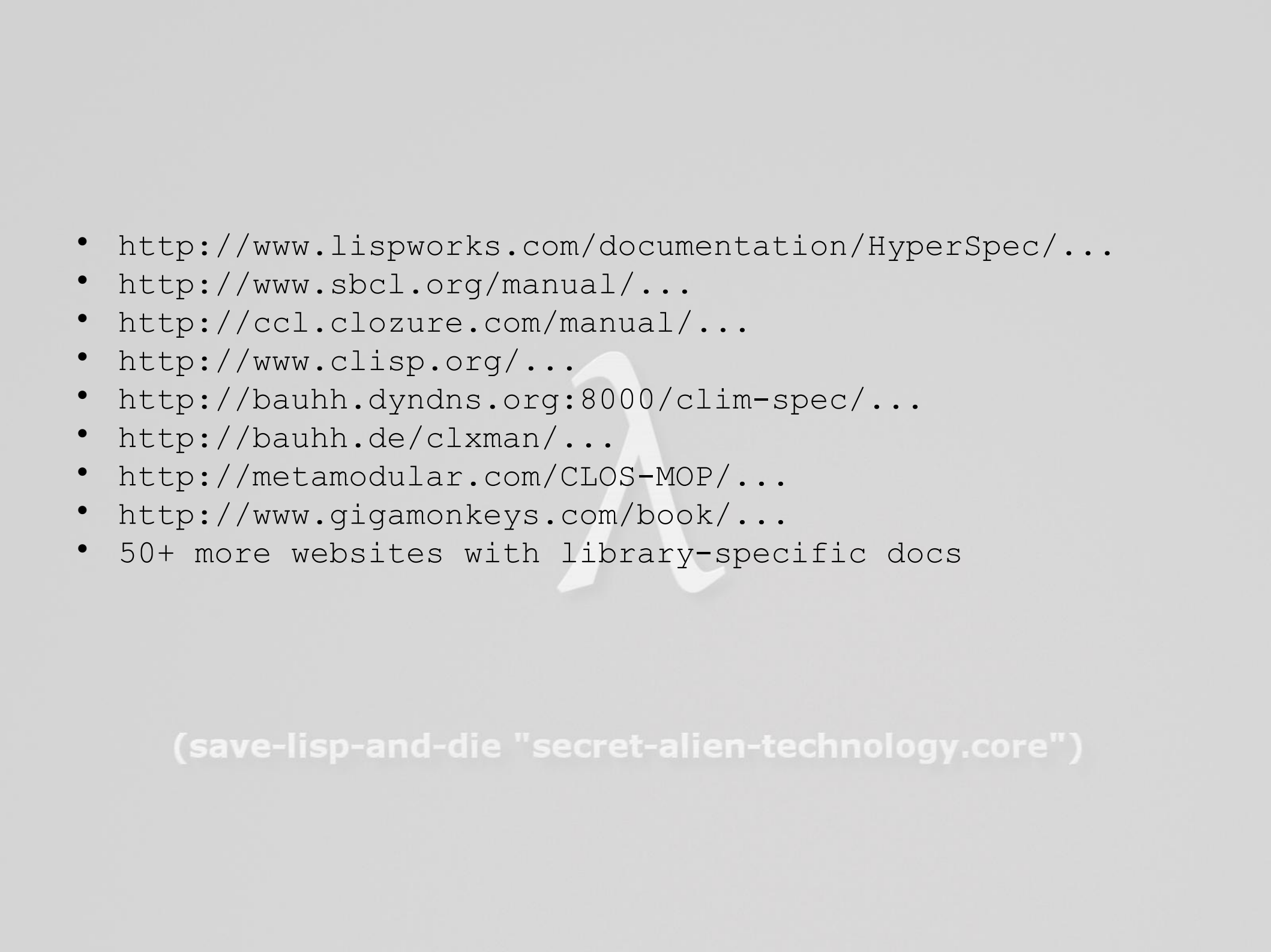
Here are some useful starting points:

A text-only version of this cover sheet is available.


Google
clhs array
Wszystko
Grafika
Mapy
Filmy
Wiadomości
Więcej
Narzędzia wyszukiwania
Około 6 490 wyników (0,17 s)
CLHS: Function MAKE-ARRAY - LispWorks
www.lispworks.com/.../lw51/CLHS/.../f_mk_ar.htm ▾ Tłumaczenie strony
Syntax: make-array dimensions &key element-type initial-element initial-contents adjustable fill-pointer displaced-to displaced-index-offset. => new-array.
CLHS: Section The Arrays Dictionary - LispWorks
www.lispworks.com/documentation/.../c_arrays.htm ▾ Tłumaczenie strony
15.2 The Arrays Dictionary. System Class ARRAY · Type SIMPLE-ARRAY · System Class VECTOR · Type SIMPLE-VECTOR · System Class BIT-VECTOR.
CLHS: System Class ARRAY
clhs.lisp.se/Body/t_array.htm ▾ Tłumaczenie strony
An array contains objects arranged according to a Cartesian coordinate system. An array provides mappings from a set of fixnums {i0,i1,...,ir-1} to corresponding ...
Ta strona była przez Ciebie odwiedzana.
CLHS: Function ADJUST-ARRAY
clhs.lisp.se/Body/f_adjust.htm ▾ Tłumaczenie strony
Syntax: adjust-array array new-dimensions &key element-type initial-element initial-contents fill-pointer displaced-to displaced-index-offset. => adjusted-array.

Content

- http://www.lispworks.com/documentation/HyperSpec/...
- http://www.sbcl.org/manual/...
- http://ccl.clozure.com/manual/...
- http://www.clisp.org/...
- http://bauhh.dyndns.org:8000/clim-spec/...
- http://bauhh.de/clxman/...
- http://metamodular.com/CLOS-MOP/...
- http://www.gigamonkeys.com/book/...
- 50+ more websites with library-specific docs



Fin

(save-lisp-and-die "secret-alien-technology.core")

Thanks for listening!

A Possible Solution To The State Of Common Lisp Documentation

Common Lisp UltraSpec

Michał „phoe” Herda

```
%-------------------- List Mapping --------------------


%%% ========== MAPCAR
%%% ========== MAPLIST
%%% ========== MAPC
%%% ========== MAPL
%%% ========== MAPCAN
%%% ========== MAPCON
\begincom{mapc, mapcar, mapcan, mapl, maplist, mapcon}\ftype{Function}

\label Syntax::

\DefunWithValues mapc    {function {\rest} \plus{lists}} {list-1}
\DefunWithValues mapcar  {function {\rest} \plus{lists}} {result-list}
\DefunWithValues mapcan  {function {\rest} \plus{lists}} {concatenated-results}
\DefunWithValues mapl    {function {\rest} \plus{lists}} {list-1}
\DefunWithValues maplist {function {\rest} \plus{lists}} {result-list}
\DefunWithValues mapcon  {function {\rest} \plus{lists}} {concatenated-results}

\label Arguments and Values::

\param{function}---a \term{designator} for a \term{function}
  that must take as many \term{arguments} as there are \param{lists}.

\issue{DOTTED-LIST-ARGUMENTS:CLARIFY}
\param{list}---a \term{proper list}.

\param{list-1}---the first \param{list} (which must be a \term{proper list}).
\endissue{DOTTED-LIST-ARGUMENTS:CLARIFY}
```

====== Function MAPC, MAPCAR, MAPCAN, MAPL, MAPLIST, MAPCON ======
The mapping operation involves applying //function// to successive sets of arguments in which one argument is obtained from each //[[CL:Glossary:sequence]]//. Except for **mapc** and **mapl**, the result contains the results returned by //function//. In the cases of **mapc** and **mapl**, the resulting //[[CL:Glossary:sequence]]// is //list//.

//function// is called first on all the elements with index ''0'', then on all those with index ''1'', and so on. //result-type// specifies the //[[CL:Glossary:type]]// of the resulting //[[CL:Glossary:sequence]]//. If //function// is a //[[CL:Glossary:symbol]]//, it is **[[CL:Functions:coerce]]**d to a //[[CL:Glossary:function]]// as if by **[[CL:Functions:symbol-function]]**.

mapcar operates on successive //[[CL:Glossary:element|elements]]// of the //lists//. //function// is applied to the first //[[CL:Glossary:element]]// of each //list//, then to the second //[[CL:Glossary:element]]// of each //list//, and so on. The iteration terminates when the shortest //list// runs out, and excess elements in other lists are ignored. The value returned by **mapcar** is a //[[CL:Glossary:list]]// of the results of successive calls to //function//.

mapc is like **mapcar** except that the results of applying //function// are not accumulated. The //list// argument is returned.

maplist is like **mapcar** except that //function// is applied to successive sublists of the //lists//. //function// is first applied to the //lists// themselves, and then to the //[[CL:Glossary:cdr]]// of each //list//, and then to the //[[CL:Glossary:cdr]]// of the //[[CL:Glossary:cdr]]// of each //list//, and so on.

mapl is like **maplist** except that the results of applying //function// are not accumulated: //list-1// is returned.

Function MAPC, MAPCAR, MAPCAN, MAPL, MAPLIST, MAPCON

The mapping operation involves applying *function* to successive sets of arguments in which one argument is obtained from each *sequence*. Except for **mapc** and **mapl**, the result contains the results returned by *function*. In the cases of **mapc** and **mapl**, the resulting *sequence* is *list*.

function is called first on all the elements with index `0`, then on all those with index `1`, and so on. *result-type* specifies the *type* of the resulting *sequence*. If *function* is a *symbol*, it is **coerce**d to a *function* as if by **symbol-function**.

mapcar operates on successive *elements* of the *lists*. *function* is applied to the first *element* of each *list*, then to the second *element* of each *list*, and so on. The iteration terminates when the shortest *list* runs out, and excess elements in other lists are ignored. The value returned by **mapcar** is a *list* of the results of successive calls to *function*.

mapc is like **mapcar** except that the results of applying *function* are not accumulated. The *list* argument is returned.

maplist is like **mapcar** except that *function* is applied to successive sublists of the *lists*. *function* is first applied to the *lists* themselves, and then to the *cdr* of each *list*, and then to the *cdr* of the *cdr* of each *list*, and so on.

mapl is like **maplist** except that the results of applying *function* are not accumulated; *list-1* is returned.

mapcan and **mapcon** are like **mapcar** and **maplist** respectively, except that the results of applying *function* are combined into a *list* by the use of **nconc** rather than **list**. That is,

```
(mapcon f x1 … xn) ≡ (apply #'nconc (maplist f x1 … xn))
```

and similarly for the relationship between **mapcan** and **mapcar**.

Syntax

- **mapc** *function* `&rest` *lists+* → *list-1*
- **mapcar** *function* `&rest` *lists+* → *result-list*

Live demo/manifesto:

http://phoe.tymoon.eu/clus/

(save-lisp-and-die "secret-alien-technology.core")

Thanks to Shinmera for the hosting!

What am I aiming for?

- Editable
- Complete
- Downloadable
- Mirrorable/Clonable
- Versioned
- Modular
- Updatable
- Portable
- Unified
- Community-based

(save-lisp-and-die "secret-alien-technology.core")

The Actual End

(save-lisp-and-die "secret-alien-technology.core")

Thanks for listening!

- 01 ILLITHID - *Mark Evenson*
- 02 Sugaring Lisp for the 21st Century - *Vsevolod Dyomkin*
- 03 Common Lisp UltraSpec - *Michał Herda*
- **04 PRIMSMA - *Philip Gagner***
- 05 Managing Packages - *James Anderson*
- 06 CEPL Reaches Beta - *Chris Bagley*
- 07 QueryFS: a Solution Looking for More Problems - *Michael Raskin*
- 08 LISP/c - *Michał Herda (on behalf of Jonathan Baca)*
- 09 Electricity is Orange - *Devon McCullough*

PRIMSMA

- PRISMA is a new AI project of SCG, supported by Thompson Reuters' EIKON and ELEKTRON.

- Thompson Reuters is the largest non-military public data harvester in the world*.

- T-R data includes shipping, insurance, energy, scientific papers & data, most only for special use.

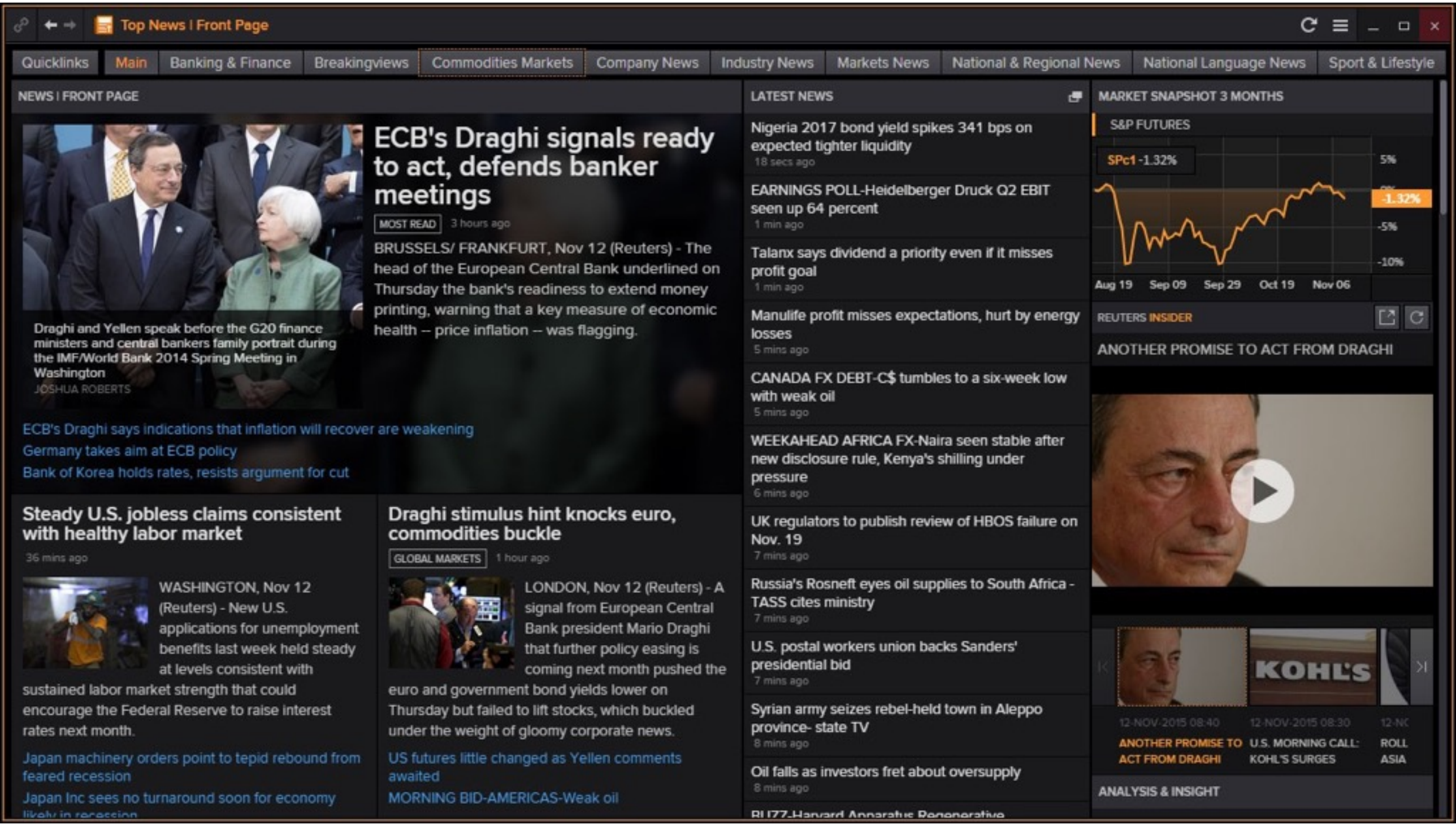
*Not counting CERN and other specialized harvesters

Why do my slides have a cute dragon on them?

- C’mon, doesn’t that answer itself?*

(And because the dragon is a symbol of Romania)

T-R EIKON “SCREEN”



(inject AI T-R-EIKON)

- EIKON currently can access only a fraction of available data
- PRISMA will add new data sources, but more importantly, new ways of viewing the new and old data
- What primary programming languages?
    - Javascript
    - C#
    - And Lisp for all the hard stuff

Changing the EIKON market

- T-R views EIKON mainly as a financial analyst tool
- We view EIKON as a platform for delivering economic and political analysis
- T-R has agreed to market our software on EIKON

Changing the EIKON market

- EIKON has ~300,000 subscribers, mostly financial and governments
- We want to dramatically expand their market into political and economic analysis
- We want to distribute human analysis and replace the analysts with Lisp code
- T-R has agreed to market our software on EIKON

Why Bucharest?

- It’s a great city!
- Located in a country whose GDP is increasing, and debt shrinking as %GDP, top 1/3 in most economic indicators
- Lots of technology and a history of innovation*

*The jet aircraft was invented in Romania, but not implemented in Lisp

Why Romania?

- Cost of living is low, yet much to see and do
- Situated between Western Europe and Middle East
- We have 2 teams of human analysts in Romania now
- Romania built a Lisp Machine in 1984 (Professor George Stefan is one of our advisors)

DIALISP - A LISP MACHINE

G.Stefan
A.Paun
V.Bistriceanu
A.Birnbaum

Functional Electronics Laboratory
Polytechnical Institute of Bucharest
ROMANIA

A B S T R A C T

High performance facilities to interpret LISP represent an ever increasing request even for minis.

This paper presents a LISP hardware structure conceived to be implemented in a general purpose mini system called DIAGRAM.

The LISP structure had to be adapted to the system technological requirements and size.

The data structure and the instruction set concerning the basic machine are also presented.

1.Introduction

The system comprising the LISP machine is shown in Fig.1, where:

- IOM is a microcomputer controlling the system input-output devices;
- MPM1 is a minicomputer on a PCB, operating as the system central unit, running high level languages (Fortran, Basic, a.s.o.). It operates with a general purpose arithmetic processor;
- MPM0 is a physical structure identical with MPM1, controlling the alphanumeric and graphic display on a black and white or color CRT monitor. It can operate with a numerical processor specialized in bi- and tridimensional graphic transformations;
- DIALISP, the topic of this paper, is the LISP hardware interpreter.

2.General structure of LISP Hardware Interpreter (DIALISP)

2.1.Structural Options

The acces time of the available memory devices used in high capacity memory arrays is 300-500 ns.

Using TTL devices, processing structures (RALU, CROM,...) having cycle time between 150 and 300 ns can be obtained.

Hence, using a cache memory the efficiency may be rather poor even when the processes associated to the LISP interpretation frequently access the memory.

Hardware facilities offered by bit-slices controlled by a stack state-machine (SSM) have been used to optimize DIALISP cost and size. Thus the whole structure is built on a single PCB, but the operating speed is

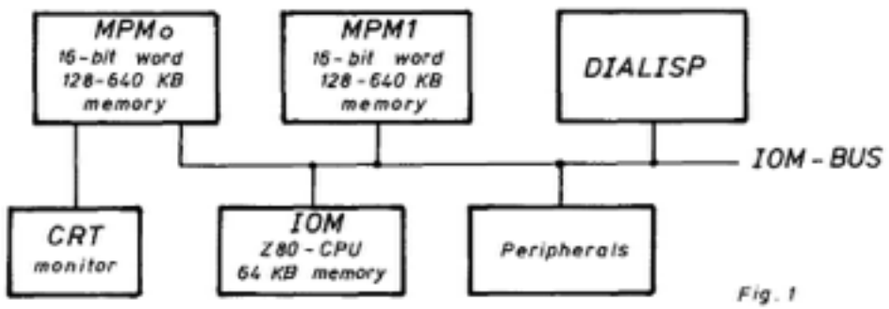


Fig. 1

limited due to the small number of fast registers and to the use of arithmetic comparing functions instead of logic ones, as in a dedicated structure.

Using a SSM to control the structure instead of a typical CROM configuration permits higher speed and the implementation of a large micro-stack.

2.2.Duality

An important option for DIALISP takes

What Positions?

- An Experienced Lisp Programmer willing to share knowledge
- Lisp Programmers who want to do real AI with Text Processing of Big Data
- Enthusiastic programmers willing to learn Lisp

What Else?

- Contact me at gagner@schloerconsulting.com
- Have your friends contact me too (if they're Lispers)
- Salaries, bonuses, etc. are negotiable

- 01 ILLITHID - *Mark Evenson*
- 02 Sugaring Lisp for the 21st Century - *Vsevolod Dyomkin*
- 03 Common Lisp UltraSpec - *Michał Herda*
- 04 PRIMSMA - *Philip Gagner*
- **05 Managing Packages - *James Anderson***
- **06 CEPL Reaches Beta - *Chris Bagley***
- **07 QueryFS: a Solution Looking for More Problems - *Michael Raskin***
- **08 LISP/c - *Michał Herda (on behalf of Jonathan Baca)***
- **09 Electricity is Orange - *Devon McCullough***