Lightning Talks I

ELS 2017

- **01. First-Class Implementations - *François-René Rideau***
- 02. Reflectable CL Functions Using the MOP - *Jim newton*
- 03. Erlangen: Async Distributed Msg Passing for CCL - *Max Rottenkolber*
- 04. A CL Grapher for Implications Between Axioms - *Ioanna M. Dimitriou*
- 05. cl-jupyter: Lisp-Powered Jupyter Notebooks - *Frédéric Peschanski*
- 06. Trial, a New CL Game Engine - *Nicolas Hafner*

Meta-Implementation Protocol

Semantics + Reflection = First-Class Implementations

Turn your Lisp into a Meta-Platform

François-René Rideau, *TUNES Project*

Lightning Talk at the European Lisp Symposium, 2017-04-03

http://fare.tunes.org/files/cs/fci-els2017.pdf

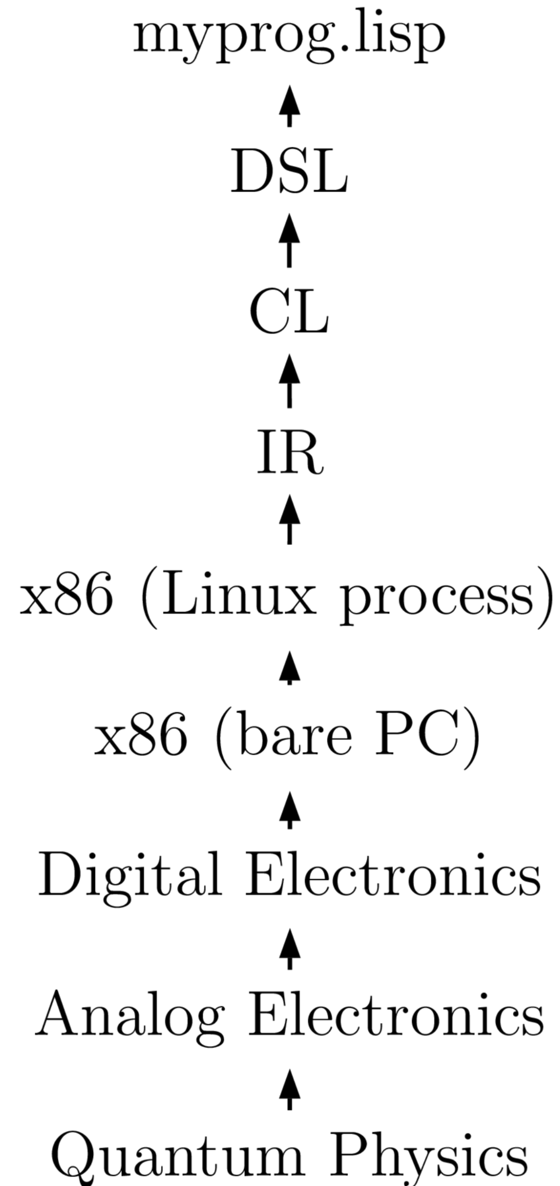
Teaser for my 2017-03-21 presentation at Lisp NYC

Basic Intuitions

Good programmers can mentally zoom in and out of levels of abstraction

Interesting theorems allow you to change your perspective on existing objects

Semantic Tower



Navigating, not mere debugging

Debugging

Local program state only

Only recover one level of abstraction

One way fixed magic operation

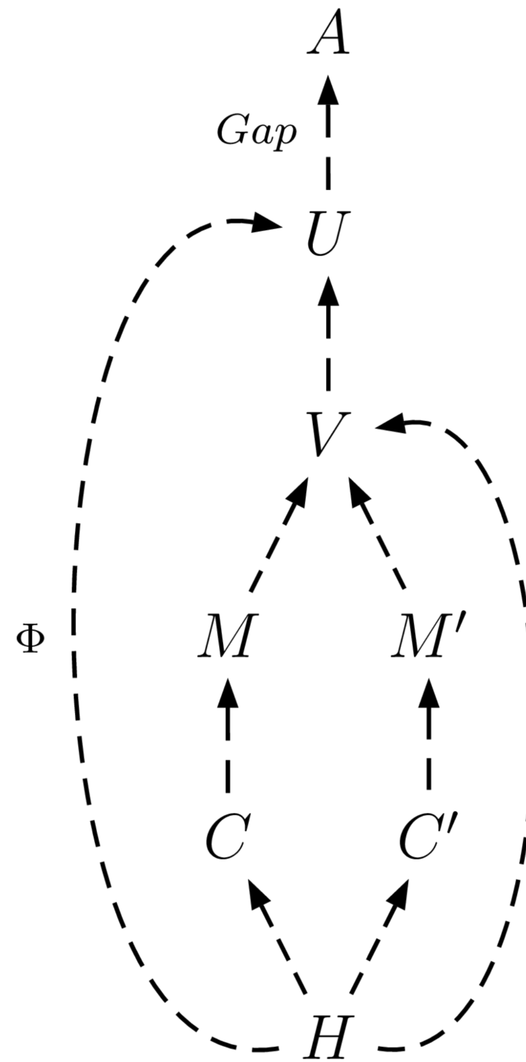
Navigating

Recurse to complete program state

Compose to recover any level you like

First-class operation both ways

Migration



When your hammer is Migration...

Process Migration

Garbage Collection

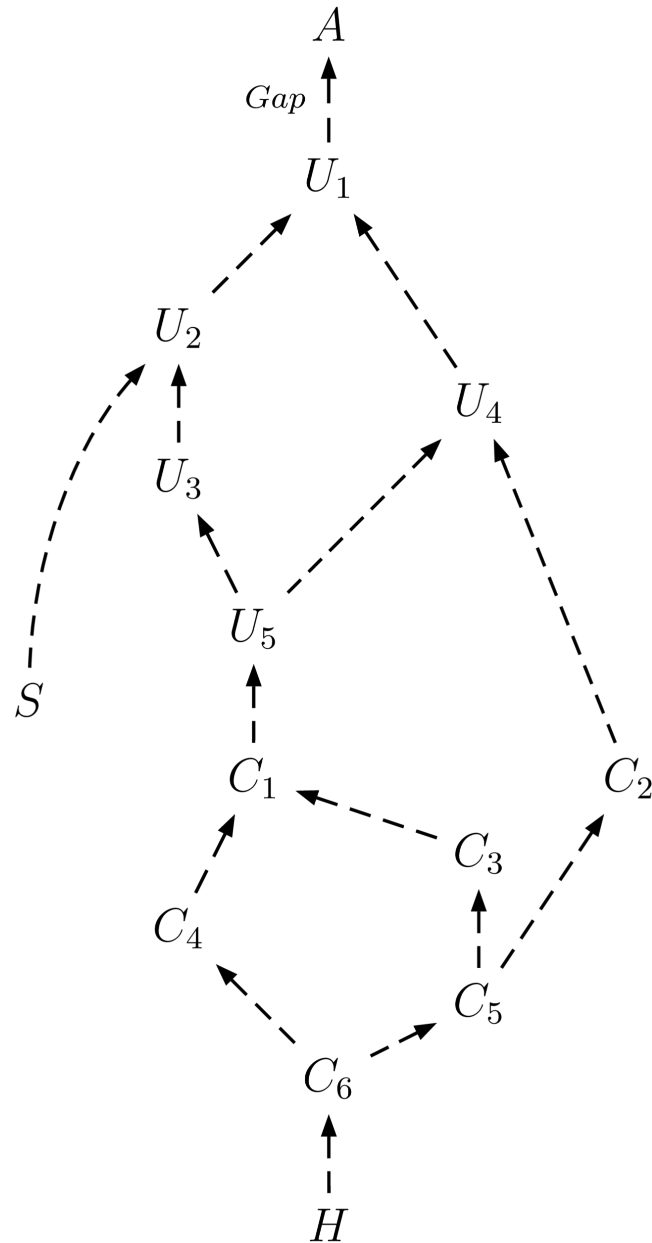
Zero Copy Routing

Dynamic Configuration

JIT Compilation

etc.

Semantic Towers need not be linear!



New Insights on...

Computation Semantics

Compilation

Semantics-preserving transformations

Aspect-Oriented Programming

Code Instrumentation

Virtualization

Computational Reflection

Software Architecture

Security

First-Class Implementations

Formalizing Implementations: Categories!

Observability: Neglected key concept — safe points

First-Class Implementations via Protocol Extraction

Explore the Semantic Tower — at runtime!

Principled Reflection: Migration

Natural Transformations generalize Instrumentation

Reflective Architecture: 3D Towers

Social Implications: Platforms, not Applications

Challenge

Put the "MIP" in your Lisp

Let’s change software architecture!

Thank you

My blog: *Houyhnhnm Computing*

http://ngnghm.github.io/

- 01. First-Class Implementations - *François-René Rideau*
- **02. Reflectable CL Functions Using the MOP - *Jim newton***
- 03. Erlangen: Async Distributed Msg Passing for CCL - *Max Rottenkolber*
- 04. A CL Grapher for Implications Between Axioms - *Ioanna M. Dimitriou*
- 05. cl-jupyter: Lisp-Powered Jupyter Notebooks - *Frédéric Peschanski*
- 06. Trial, a New CL Game Engine - *Nicolas Hafner*

Enlightening lightning talk
ELS 2017

Reflectable CL functions

Using the MOP

Too many functions. I get confused, need help debugging.

```
(defmethod map-pxls ((p-pxl-iters parallel-pxl-iters)
                     &key unary aggregate
                       (mk-unary (lambda () unary)))
  (let ((0-ary-functions
          (mapcar (lambda (p-pxl-iter
                           &aux (unary (funcall mk-unary)))
                    (lambda ()
                      (map-pxls p-pxl-iter :unary unary)))
                  (pxl-iters p-pxl-iters))))
  (lparallel:preduce (lambda (v f)
                       (funcall aggregate v (funcall f)))
                     0-ary-functions)))
```

Declaring function types doesn't help. Compiler ignores content.

```
(defmethod map-pxls ((p-pxl-iters parallel-pxl-iters)
                     &key unary aggregate
                       (mk-unary (lambda () unary)))

  (declare (type (function (t) t) unary)
           (type (function () (function (t) t)) mk-unary)
           (type (function (t t) t) aggregate))

  (let ((0-ary-functions
          (mapcar (lambda (p-pxl-iter
                           &aux (unary (funcall mk-unary)))
                    (lambda ()
                      (map-pxls p-pxl-iter :unary unary)))
                  (pxl-iters p-pxl-iters))))
  ...
```

Run-time function assertion fails.

```
(defmethod map-pxls ((p-pxl-iters parallel-pxl-iters)
                     &key unary aggregate
                       (mk-unary (lambda () unary)))
  (let ((0-ary-functions
          (mapcar (lambda (p-pxl-iter
                            &aux (unary (funcall mk-unary)))
                    (assert (typep mk-unary '(function () t)))
                    (assert (typep unary '(function (t) t)))
                    (lambda ()
                      (map-pxls p-pxl-iter :unary unary)))
                  (pxl-iters p-pxl-iters))))
  (lparallel:preduce (lambda (v f)
                       (funcall aggregate v (funcall f)))
                     0-ary-functions)))
```

CL specification of *Function* TYPEP

- *An error of type* error *is signaled if type-specifier is* `values`*, or a type specifier list whose first element is either* function *or* `values`*.*

- `(assert (typep unary ‘(function (t) t)))`

Define class with :metaclass `closer-mop::funcallable-standard-class`

```
(defclass reflectable-function ()
  ((function :initarg :function :type function)
   (body :initarg :body)
   (lambda-list :initarg :lambda-list))
  (:metaclass closer-mop:funcallable-standard-class))

(defmethod initialize-instance
     :after ((self reflectable-function) &key)
  (closer-mop:set-funcallable-instance-function
     self (slot-value self 'function)))

(defmacro refl-lambda (lambda-list &rest body)
  `(make-instance 'reflectable-function
                  :function (lambda ,lambda-list ,@body)
                  :body ',body
                  :lambda-list ',lambda-list))
```

Define a type intersecting the class

```
(defun refl-2-ary (obj)
  (and (typep obj 'reflectable-function)
       (= 2 (length (slot-value obj 'lambda-list)))))

(deftype refl-2-ary ()
  `(and reflectable-function
        (satisfies refl-2-ary)))

(map-pxls p-iters
          :unary (refl-lambda (pxl)
                   (red pxl))
          :aggregate (refl-lambda (v2 v2)
                       (max v1 v2)))
```

Declare away! Assert away!

```
(defmethod map-pxls ((p-pxl-iters parallel-pxl-iters)
                     &key unary aggregate
                       (mk-unary (refl-lambda () unary)))
  (declare (type refl-1-ary unary)
           (type refl-0-ary mk-unary)
           (type refl-2-ary aggregate))
    (let ((0-ary-functions
            (mapcar (lambda (p-pxl-iter
                              &aux (unary (funcall mk-unary)))
                      (assert (typep unary ‘refl-1-ary))
                      (refl-lambda ()
                        (map-pxls p-pxl-iter :unary unary)))
                    (pxl-iters p-pxl-iters))))
    (lparallel:preduce (lambda (v f)
                         (declare (type refl-0-ary f))
                         (funcall aggregate v (funcall f)))
                       0-ary-functions)))
```

Acknowledgement: Thanks to Pascal Costanza for CLOSER-TO-MOP

Questions?

- 01. First-Class Implementations - *François-René Rideau*
- 02. Reflectable CL Functions Using the MOP - *Jim newton*
- **03. Erlangen: Async Distributed Msg Passing for CCL -** ***Max Rottenkolber***
- 04. A CL Grapher for Implications Between Axioms - *Ioanna M. Dimitriou*
- 05. cl-jupyter: Lisp-Powered Jupyter Notebooks - *Frédéric Peschanski*
- 06. Trial, a New CL Game Engine - *Nicolas Hafner*

Erlangen: asynchronous, distributed message passing for Clozure CL (lightning talk for ELS 2017)

Max Rottenkolber <max@mr.gy>

Saturday, 1 April 2017

Erlangen

- asynchronous message passing
- reliable (supervision trees)
- inherently distributed
- inspired by Erlang, yet very different

“You can’t do that, it won’t be like Erlang!”

- uses native OS threads (can’t have 10,000)
- does not enforce strict isolation (within the boundary of a node)
- Common Lisp is synchronous (blocking I/O)

…there are also many reasons to prefer CL/Erlangen over Erlang

- for instance, Erlangen’s message queues are bounded and never leak memory

I did it, its amazing: lessons learned

- *uses native OS threads*

even if processes are heavy, supervision trees are worth their weight in gold

- *does not enforce strict isolation*

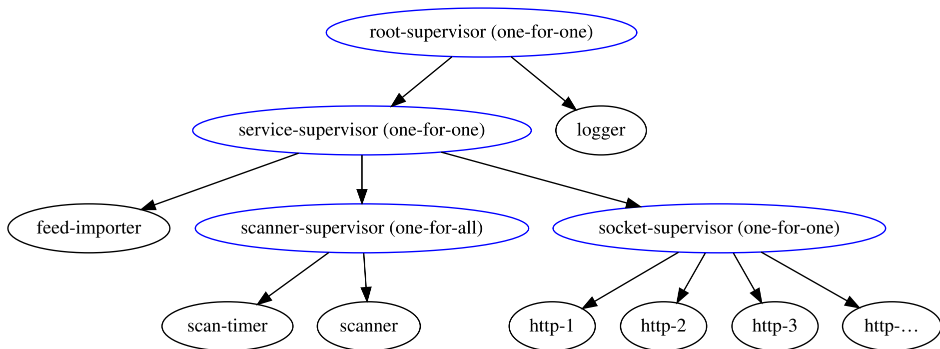
just don't modify objects you send (send = free)

- *Common Lisp is synchronous (blocking I/O)*

tempted to look at asynchronous I/O (IOLib): complete dead end, no tasteful way to write code

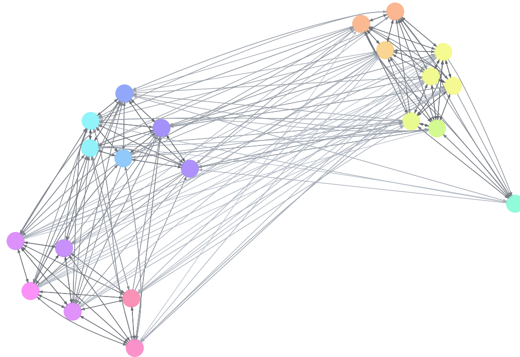
blocking is fine as long as you always supply a timeout

Write reliable services in Common Lisp



Supervision tree of a real-life application (processes can be spread on remote nodes)

Research: explore distributed designs



live visualization of 300 SLOC Kademlia DHT implementation on Erlangen

Numbers (on Intel Xeon E31245 3.30GHz)

- 5us for message delivery, 10x more than erts 7.3 (that's not even bad, actually)
- up to 2 million messages per second
- you can have 1,000 processes easily

...no serious optimization work done so far!

No SBCL, ... support because

- portability layers are boring
- portability layers are boring
- portability layers are boring
- I won't write one *or* maintain one
- Erlangen already includes modifications to CCL
- ...maybe I want to swap out CCL's threading, GC, I/O, ... implementations?
- `erlangen-kernel` is distributed as an executable that includes Quicklisp

Hack it!

- `github.com/eugeneia/erlangen` AGPL-3.0
- branch, fork all you like
- nothing is set in stone yet, wild ideas welcome
- mail me at `max@mr.gy`

ⓘⓝⓣⓔⓡ
ⓢⓣⓔⓛⓛⓐⓡ

- 01. First-Class Implementations - *François-René Rideau*
- 02. Reflectable CL Functions Using the MOP - *Jim newton*
- 03. Erlangen: Async Distributed Msg Passing for CCL - *Max Rottenkolber*
- **04. A CL Grapher for Implications Between Axioms - *Ioanna M. Dimitriou***
- 05. cl-jupyter: Lisp-Powered Jupyter Notebooks - *Frédéric Peschanski*
- 06. Trial, a New CL Game Engine - *Nicolas Hafner*

cgraph.inters.co

graphing implications of classical logic

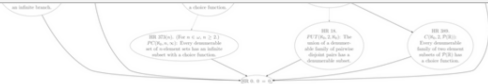
Ioanna M. Dimitriou

European Lisp Symposium 2017 - lightning session

interstellar

... axioms ....
$A \Rightarrow B$??
$A \nRightarrow B$??

cgraph.inters.co



Choiceless Grapher

Diagram creator for consequences of the axiom of choice (AC).

Enter space separated integers to get the implication diagram between the statement with these names.

List of names and statements

Enter space separated HR numbers as integers

Label style: fancy numbers

Graph these ☑ Include top and bottom node

Hover over forms and buttons for more details.
Be prepared to wait a few minutes for diagrams with over 50 forms.

Get a (pseudo) random diagram of size: Get random graph

The Choiceless Grapher can produce any size of graph of the implication and non-implication relationships between consequences of the axiom of choice, as found in the Consequences of the Axiom of Choice Project, by Prof. Paul Howard and Prof. Jean E. Rubin.

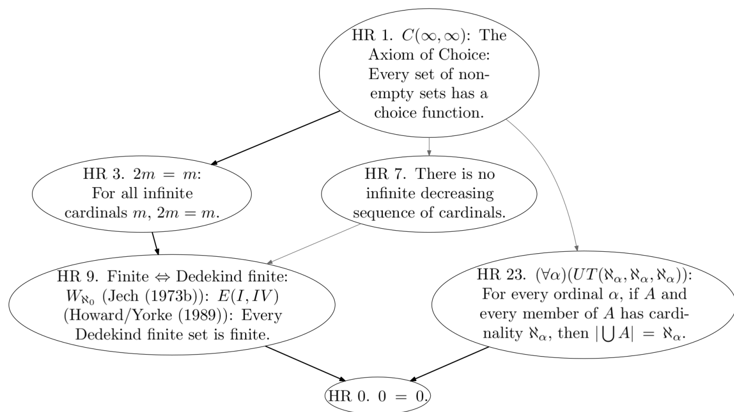
cgraph.inters.co/names-and-statements

Names and statements of the forms.

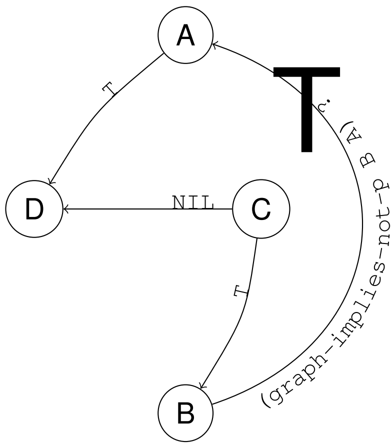
Check a form to include it in a diagram. Create this diagram using the button in the bottom of the page.

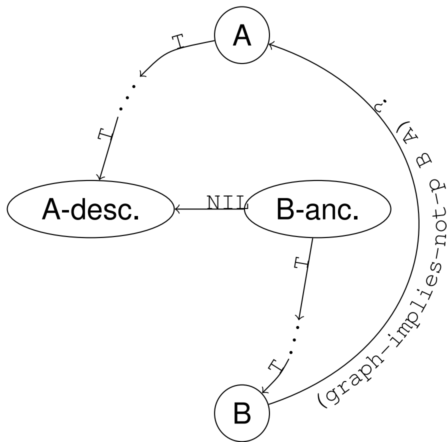
- **0.** $0 = 0$.
- **1.** $C(\infty, \infty)$: The Axiom of Choice: Every set of non-empty sets has a choice function.
- **2.** Existence of successor cardinals: For every cardinal m there is a cardinal n : $m < n$ and $(\forall p < n)(p \leq m)$.
- **3.** $2m = m$: For all infinite cardinals m , $2m = m$.
- **4.** Every infinite set is the union of some disjoint family of denumerable subsets. (Den means $\cong \aleph_0$.)
- **5.** $C(\aleph_0, \aleph_0, \mathbb{R})$: Every denumerable set of non-empty denumerable subsets of $\mathbb{R}$ has function.
- **6.** $UT(\aleph_0, \aleph_0, \aleph_0, \mathbb{R})$: The union of a denumerable family of denumerable subsets denumerable.
- **7.** There is no infinite decreasing sequence of cardinals.
- **8.** $C(\aleph_0, \infty)$: Every denumerable family of non-empty sets has a choice function.
- **9.** Finite $\Leftrightarrow$ Dedekind finite: $W_{\aleph_0}$ (see Jech 1973b); $E(I, IV)$ (see Howard/Yorke 198

inter stellar



interstellar


A
T
T
?
(graph-implies-not-p B A)
D
NIL
C
T
B



interstellar

Stack

Common Lisp
Quicklisp

backend:

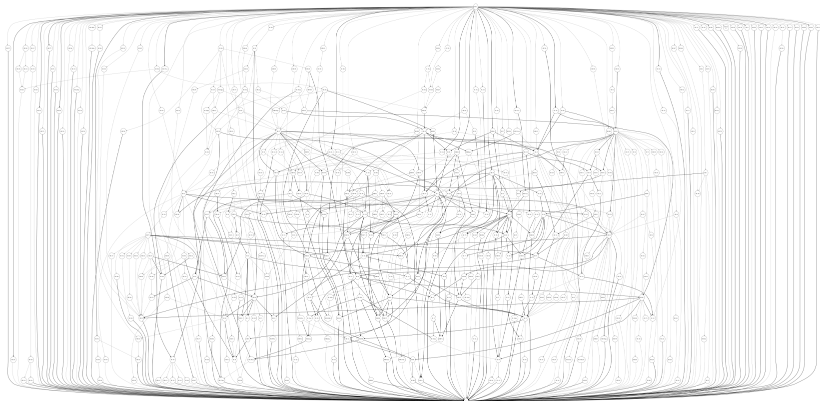
maxpc
split-sequence
bash scripts
graphviz dot + tred
external-program

frontend:

hunchentoot
html-template
a tiny JavaScript file

for more details:

cgraph.inters.co
github.com/ioannad/jeffrey
posts on boolesrings.org/ioanna
ioanna.m.dimitriou@gmail.com



the full graph

interstellar

- 01. First-Class Implementations - *François-René Rideau*
- 02. Reflectable CL Functions Using the MOP - *Jim newton*
- 03. Erlangen: Async Distributed Msg Passing for CCL - *Max Rottenkolber*
- 04. A CL Grapher for Implications Between Axioms - *Ioanna M. Dimitriou*
- **05. cl-jupyter: Lisp-Powered Jupyter Notebooks - *Frédéric Peschanski***
- 06. Trial, a New CL Game Engine - *Nicolas Hafner*

- 01. First-Class Implementations - *François-René Rideau*
- 02. Reflectable CL Functions Using the MOP - *Jim newton*
- 03. Erlangen: Async Distributed Msg Passing for CCL - *Max Rottenkolber*
- 04. A CL Grapher for Implications Between Axioms - *Ioanna M. Dimitriou*
- 05. cl-jupyter: Lisp-Powered Jupyter Notebooks - *Frédéric Peschanski*
- **06. Trial, a New CL Game Engine - *Nicolas Hafner***