Amsterdam Lisp and Scheme meetup

May 1st, 2017

Thanks for coming!

Agenda

18.45 Common lisp, good parts (Dmitry Petrov)

19.20 Stu; understanding “code is data is code” (Michael Austin)

20.00 Red language overview (Maxim Velesyuk)

- Breaks if needed
- Bar works till 20:00
- We’re looking for speakers!
- We’re looking for help!

Common Lisp

Good parts

Amsterdam Lisp & Scheme Meetup
May 1st 2017

Dmitry Petrov

Hot stuff! © Hacker news

Promises/A+

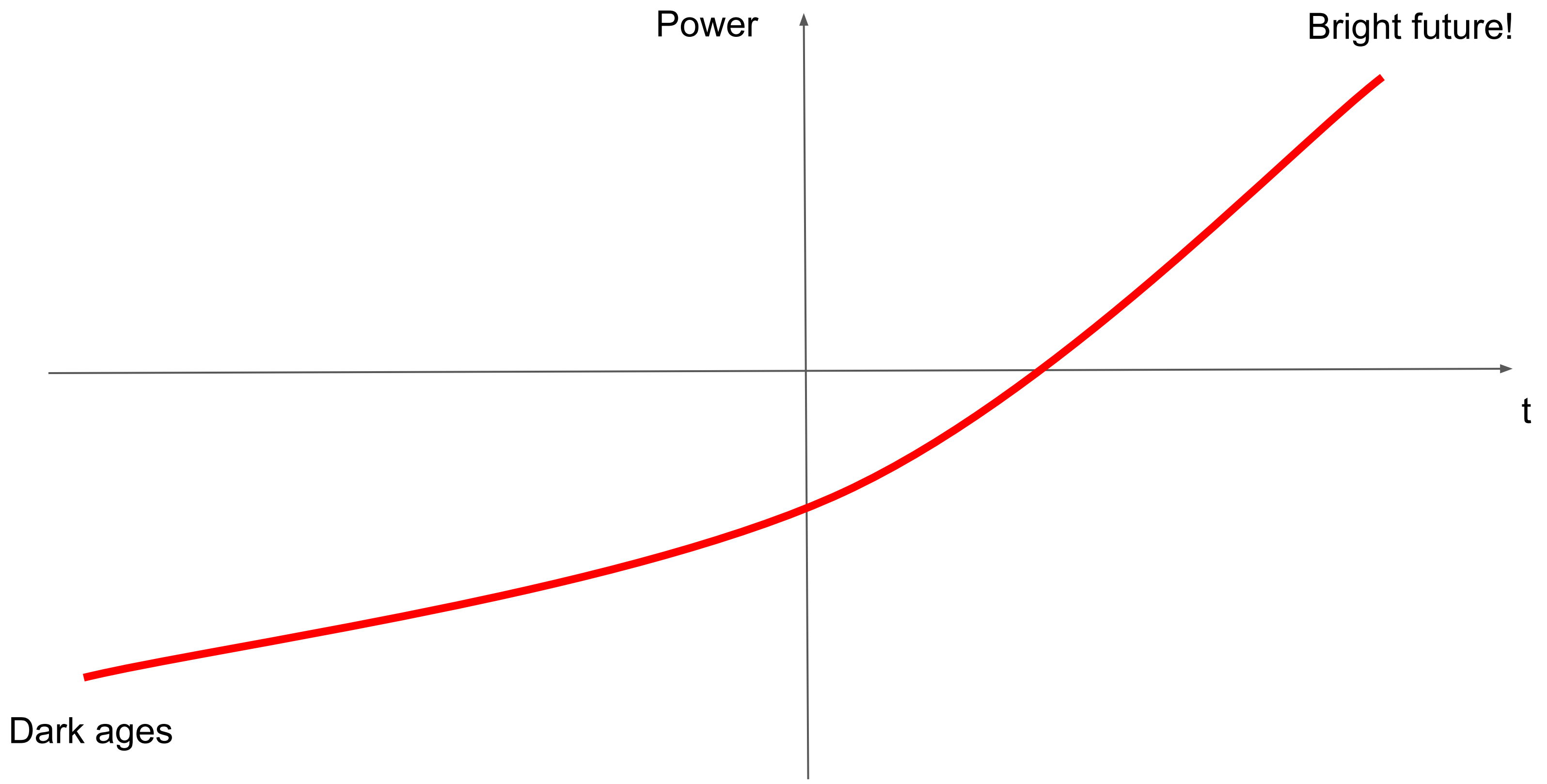
Event loop

Generators

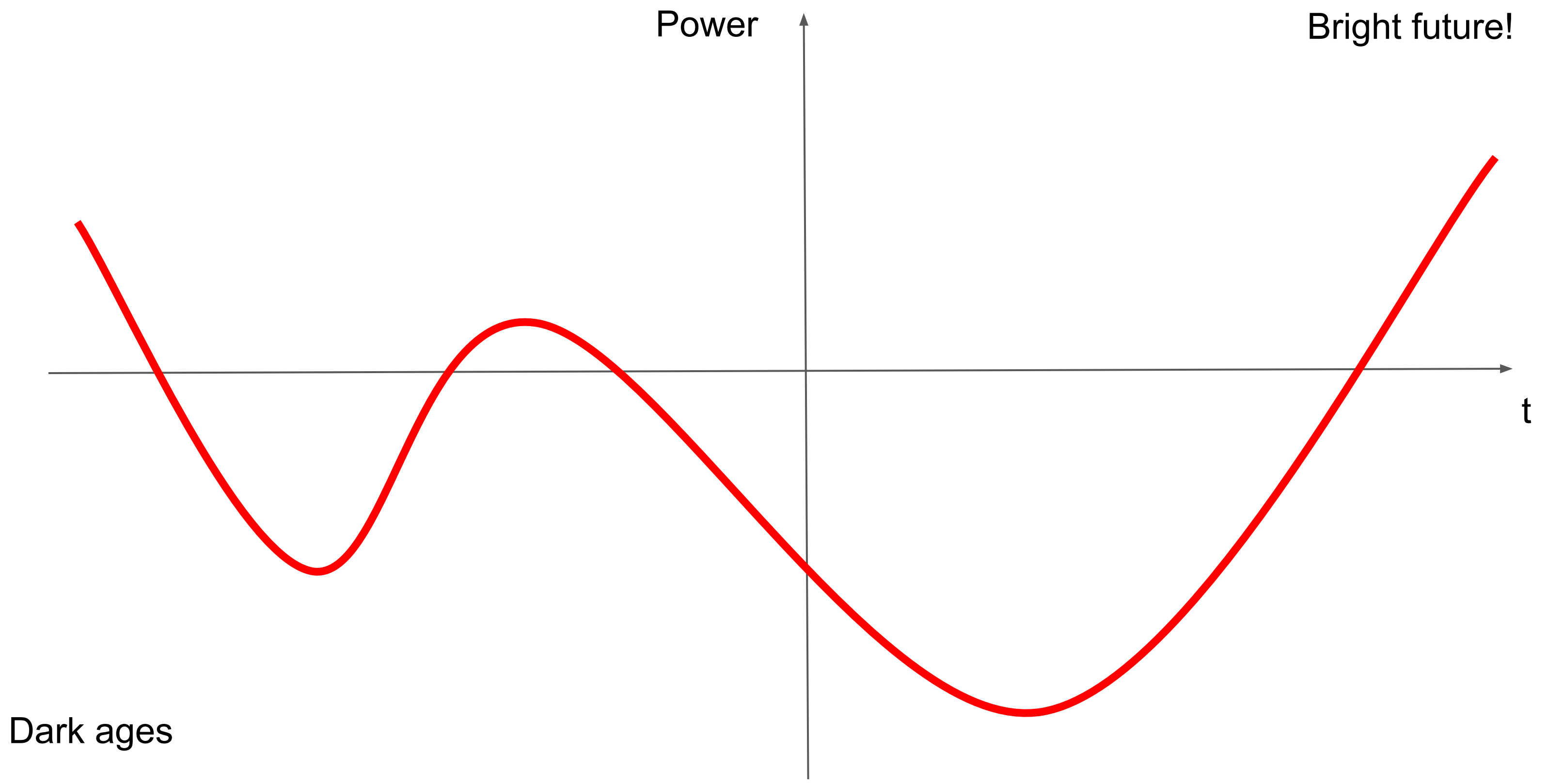
OOP

Virtual DOM

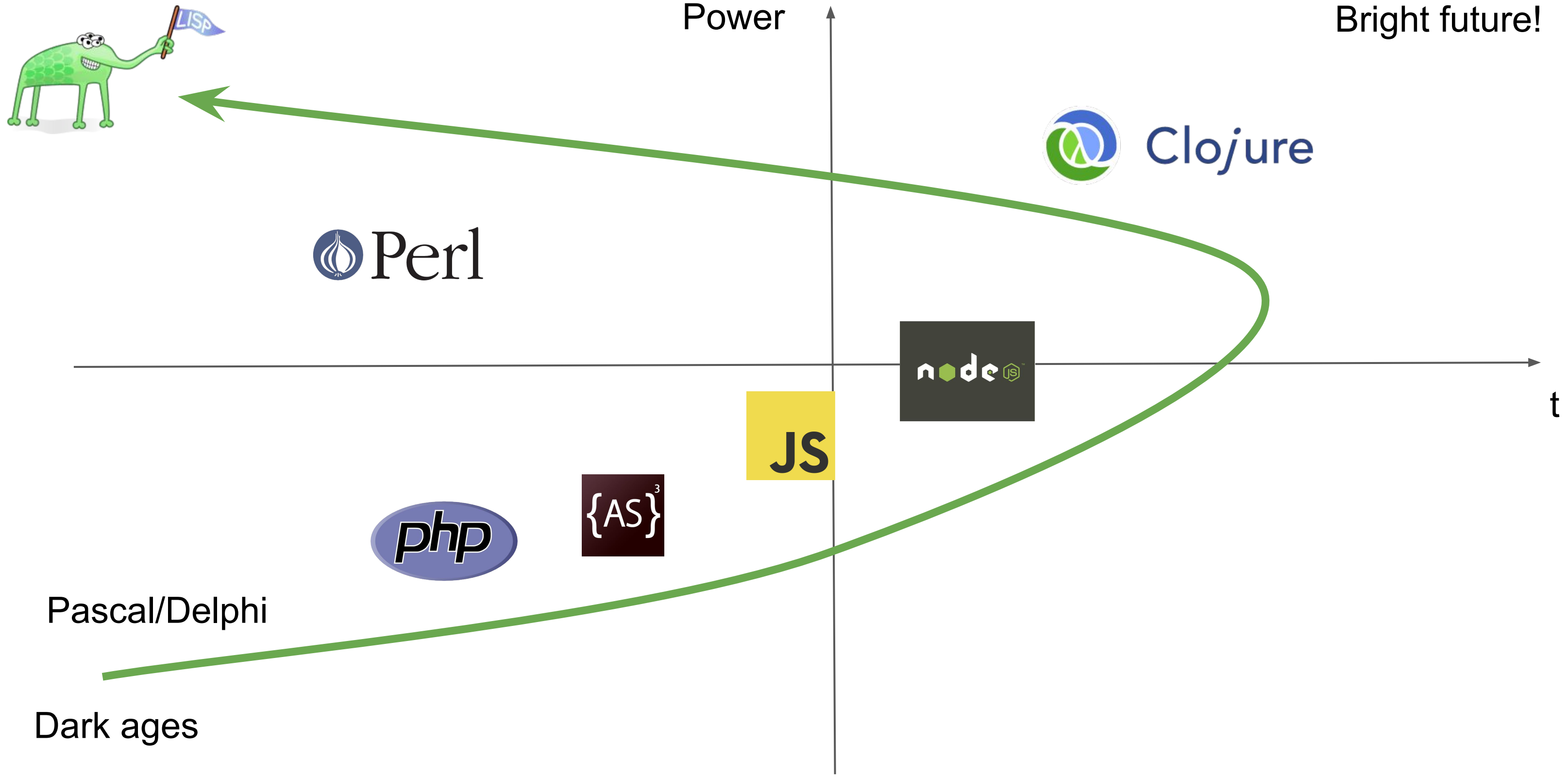
Timeline of programming languages



Timeline of programming languages



My Personal timeline of programming languages



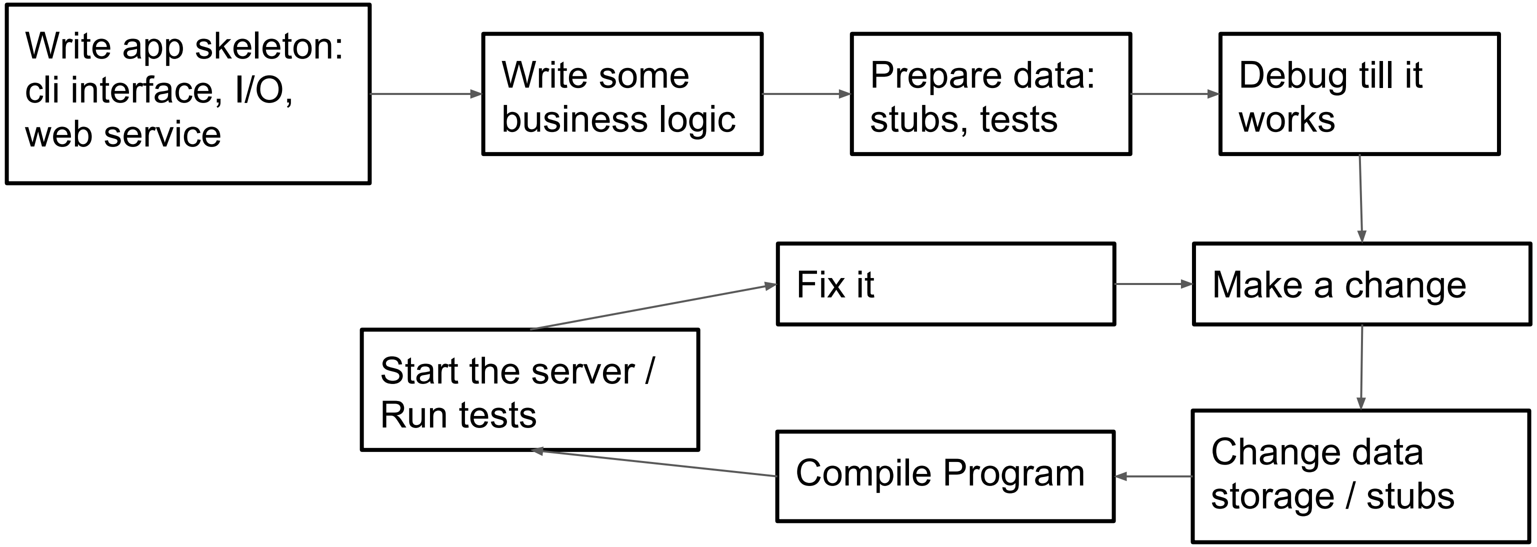
Common Lisp

Common Lisp: pillars

- Stability
- Interactive development
- Extensibility

Interactive development

Usual development loop


Write app skeleton: cli interface, I/O, web service
Write some business logic
Prepare data: stubs, tests
Debug till it works
Make a change
Change data storage / stubs
Compile Program
Start the server / Run tests
Fix it

Common lisp development loop

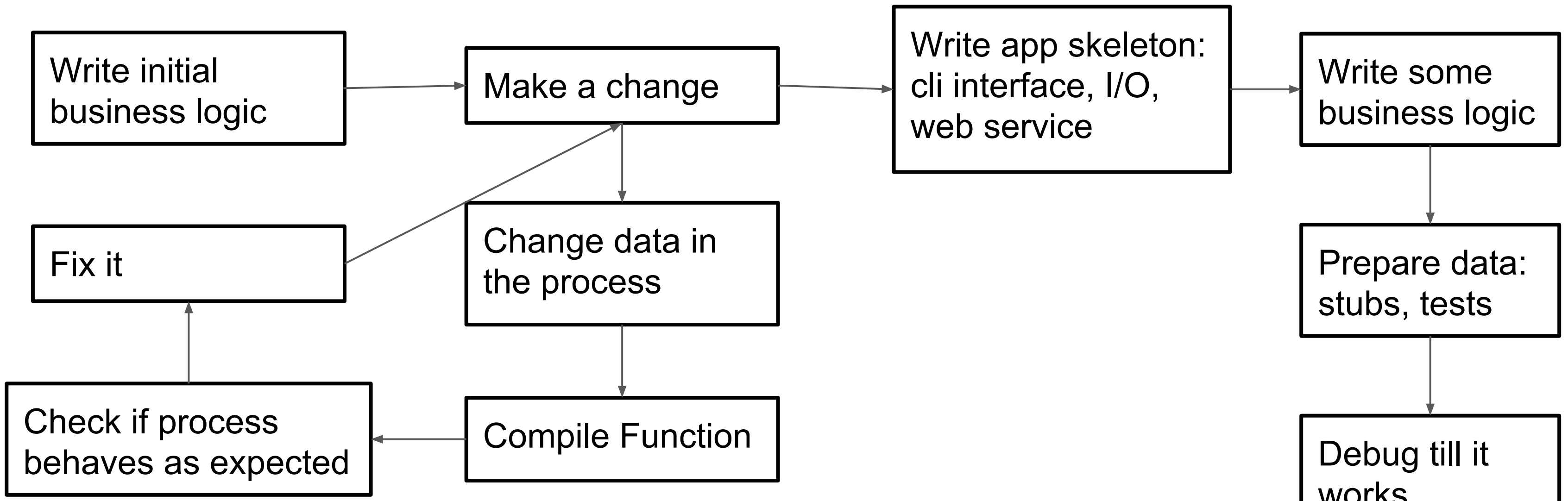

Write initial business logic
Make a change
Write app skeleton: cli interface, I/O, web service
Write some business logic
Fix it
Change data in the process
Prepare data: stubs, tests
Check if process behaves as expected
Compile Function
Debug till it works

Image based interactive development

```
(ql:quickload 'trivial-dump-core 'your-awesome-project)

(trivial-dump-core:save-executable
  "awesome_project"
  #'your-awesome-project:main)
```

Extensibility

Assignment

Most languages:

```
var variable = "value";
myObj.variable = "value";
```

Common lisp:

```
(setf <place> <form>)
```

```
(setf var "value")
```

```
(setf (aref var 2) "value")
```

```
(setf (some-field obj) "value")
```

```
(setf (my-storage 2) "value")
```

```
(setf (my-storage 2) "value")
```

Generics

```
(defgeneric print-xml (stream element &optional indent))

(defmethod print-xml (stream (element <element>) &optional (indent 0)) …)

(defmethod print-xml (stream (element <dict-article>) &optional (indent 0)) …)
```

```
(defmethod delete-post :after ((db <db>) (post <post>))
  (save-posts))
```

CLOS

```
(defclass <rectangle> ()
  ((height :initform 0.0 :initarg :height :reader height)
  (width :initform 0.0 :initarg :width :accessor width)))

(defclass <color-mixin> ()
  ((cyan :initform 0.0 :initarg :cyan)
   (magenta :initform 0.0 :initarg :magenta)
   (yellow :initform 0.0 :initarg :yellow)))

(defclass <color-rectangle> (<color-mixin> <rectangle>) ())

(defgeneric paint (x))
```

Macros

```
class Widget {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    },
    run() { /* useful stuff */ }
}

API.registerWidget("Widget", Widget)
```

Macros

```
defWidget Widget(height, width) {
        /* useful stuff */
}
```

```
(defmacro define-widget (name params &rest body)
  `(progn
     (defclass ,name () ,(expand-params params))
     (defmethod run ((widget ,name)) ,@body)
     (register-widget ,(symbol-name name) ,name)))
```

A few (lots of) messy parts

- packages
- quicklisp
- mapc* and other functions
- lacking essential apis in std lib

Where to go further

- “Practical Common Lisp”, Peter Seibel
- “The Art of the Metaobject Protocol”, Gregor Kiczales and others
- fukamachi/cl-project

  roswell/roswell

  http://lisp-lang.org/style-guide/
- quickdocs.org
- #lisp, #lispgames @freenode.net
- 40ants.com

Questions?

https://can3p.github.io