Basic sound processing in Common Lisp

Basic?

Why? Life Roadmap

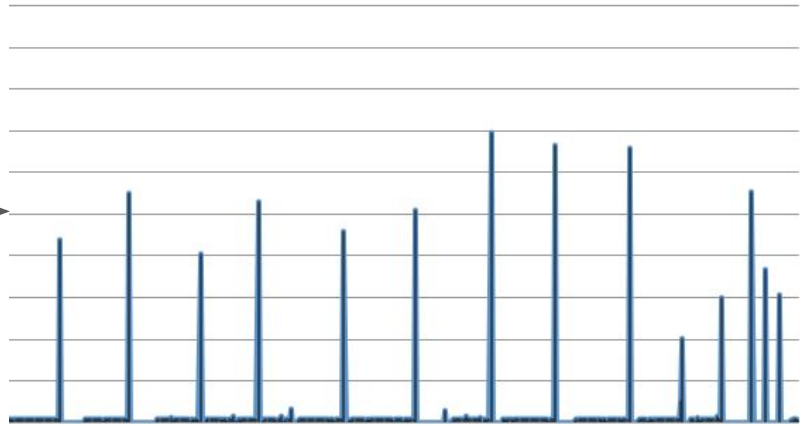
- Write something meaningful in (Common) Lisp
- Don’t do web development
- Do robots
- Decentralized social networks
- Write a simulation of office elevator
- Write simulation for port loading dock
- Automatically count jumps through the jump rope.

Lomer's

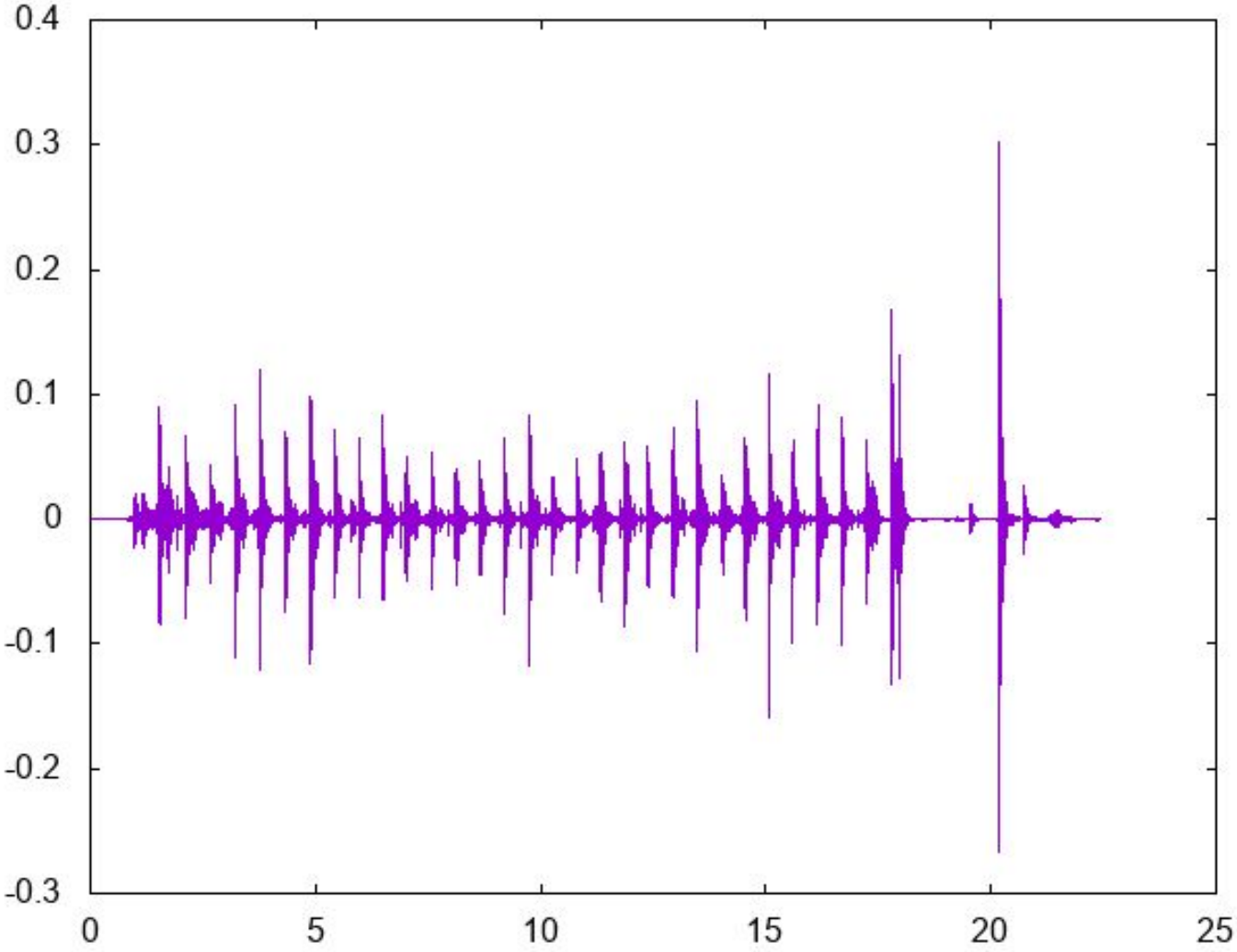
Opposite Railway Station.

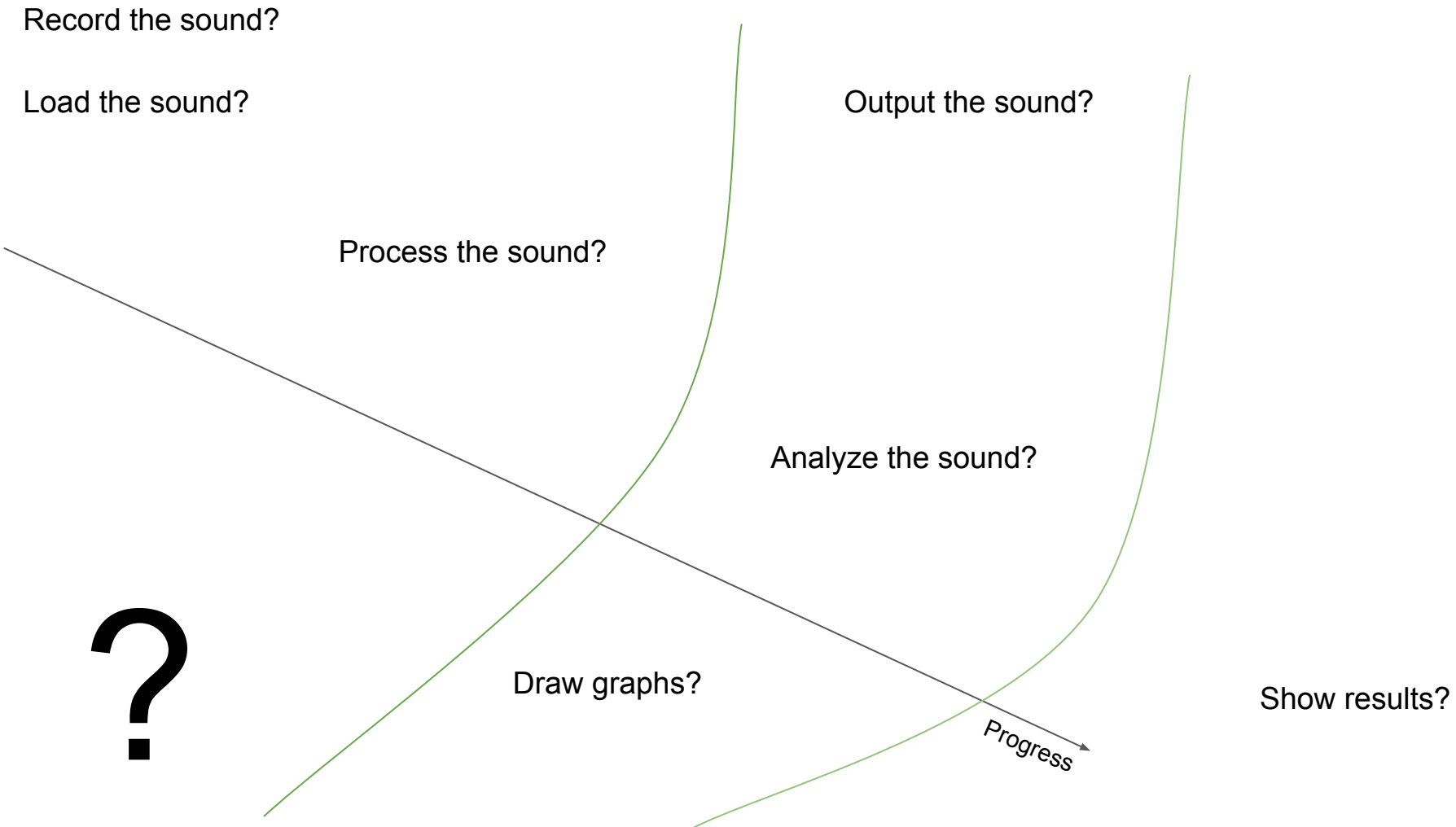
805 GEORGE STREET,
SYDNEY.

MVP



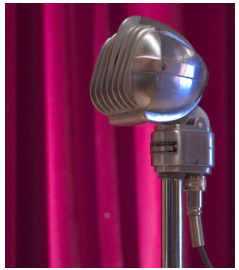
1!...2!...3!!...4!!!#!


0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0
5
10
15
20
25


Record the sound?
Load the sound?
Output the sound?
Process the sound?
Analyze the sound?
?
Draw graphs?
Show results?
Progress

github@can3p/wave-research

jumps.wav



MAGIC

Output

Input

- filonenko-mikhail/cl-portaudio

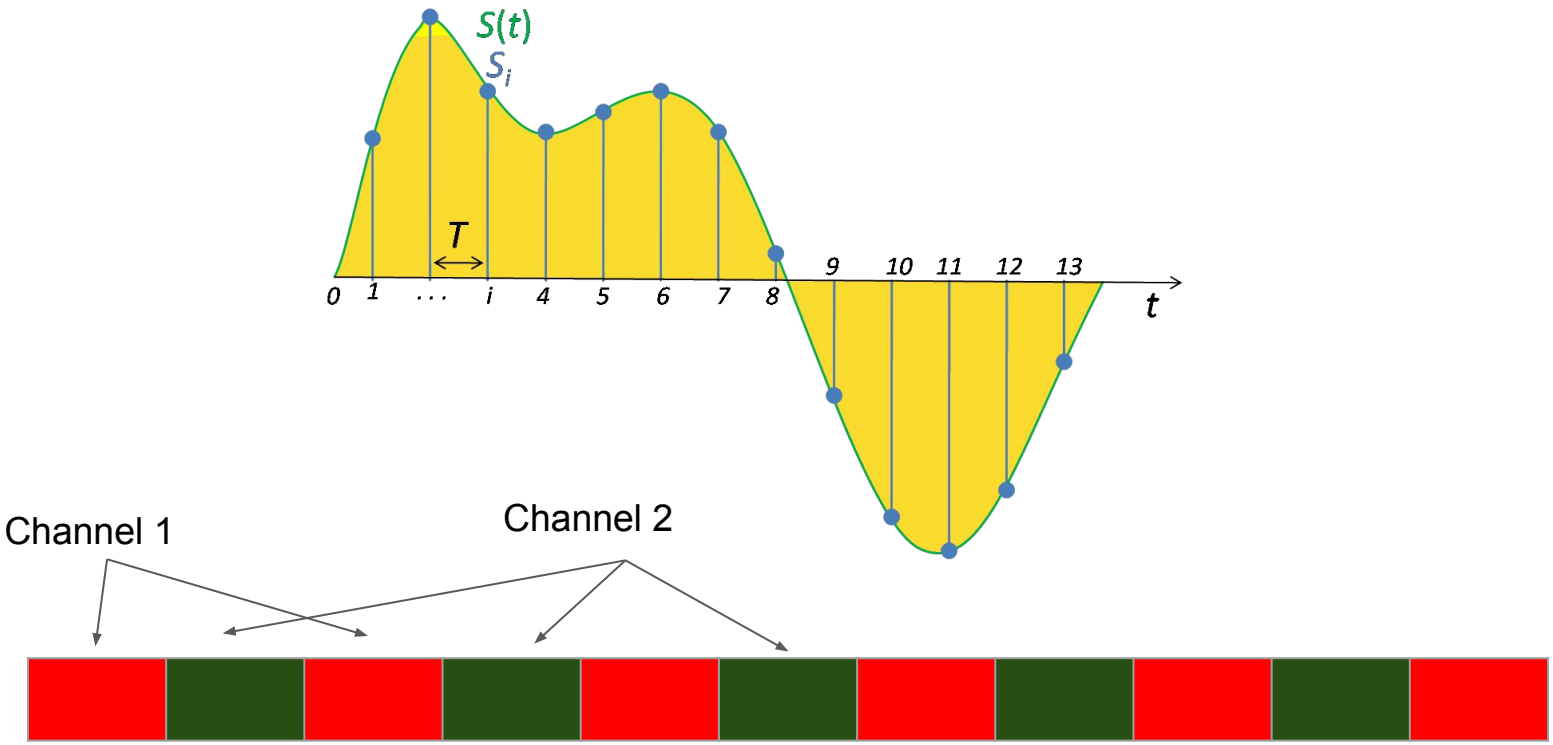
```
(with-audio
  (with-default-audio-stream (astream num-channels num-channels
                              :sample-format :float
                              :sample-rate sample-rate
                              :frames-per-buffer frames-per-buffer)
    (loop while t
          do (do-stuff (read-stream astream) 0 buffer-size)
```

Input

- RobBlackwell/cl-wav

```
(-<> source
    (cl-wav:read-wav-file <> :chunk-data-reader
                (wrap-data-chunk-data-samples-reader))
    (caddr)
    (getf :chunk-data))
```

samples = time x frequency rate x channels



Output

- filonenko-mikhail/cl-portaudio

```
(with-audio
    (with-default-audio-stream (astream num-channels num-channels
                                :sample-format :float
                                :sample-rate sample-rate
                                :frames-per-buffer frames-per-buffer)

      (for-each-buffer wave #'(lambda (buffer)
          (write-stream astream (audio-data buffer)))))
```

Graphs

SBCL + SDL =  *

Emacs + PNG = 

eazy-gnuplot

* Mac OS X

```diff
+++ b/settings.org
@@ -297,9 +297,11 @@
    :config
      (load (expand-file-name "~/.roswell/lisp/quicklisp/slime-helper.el"))
      (setq inferior-lisp-program "ros -L sbcl -Q run")
+     (setq slime-enable-evaluate-in-emacs t) ; needed to display images
      ;;(setq inferior-lisp-program "ros -L ecl -Q run")
      (setq slime-contribs '(slime-fancy
                             slime-indentation
+                            slime-media
                             slime-sbcl-exts
                             slime-scratch)))
```

github@can3p/emacs.d

```
(defun plot-png (pathname)
  (let ((filename (namestring pathname)))
    (swank:eval-in-emacs
     `(slime-media-insert-image (create-image ,filename)
                                ,filename))))

(defmethod print-object :before ((obj pathname) stream)
  (when (string= "png" (pathname-type obj))
    (plot-png obj)))
```

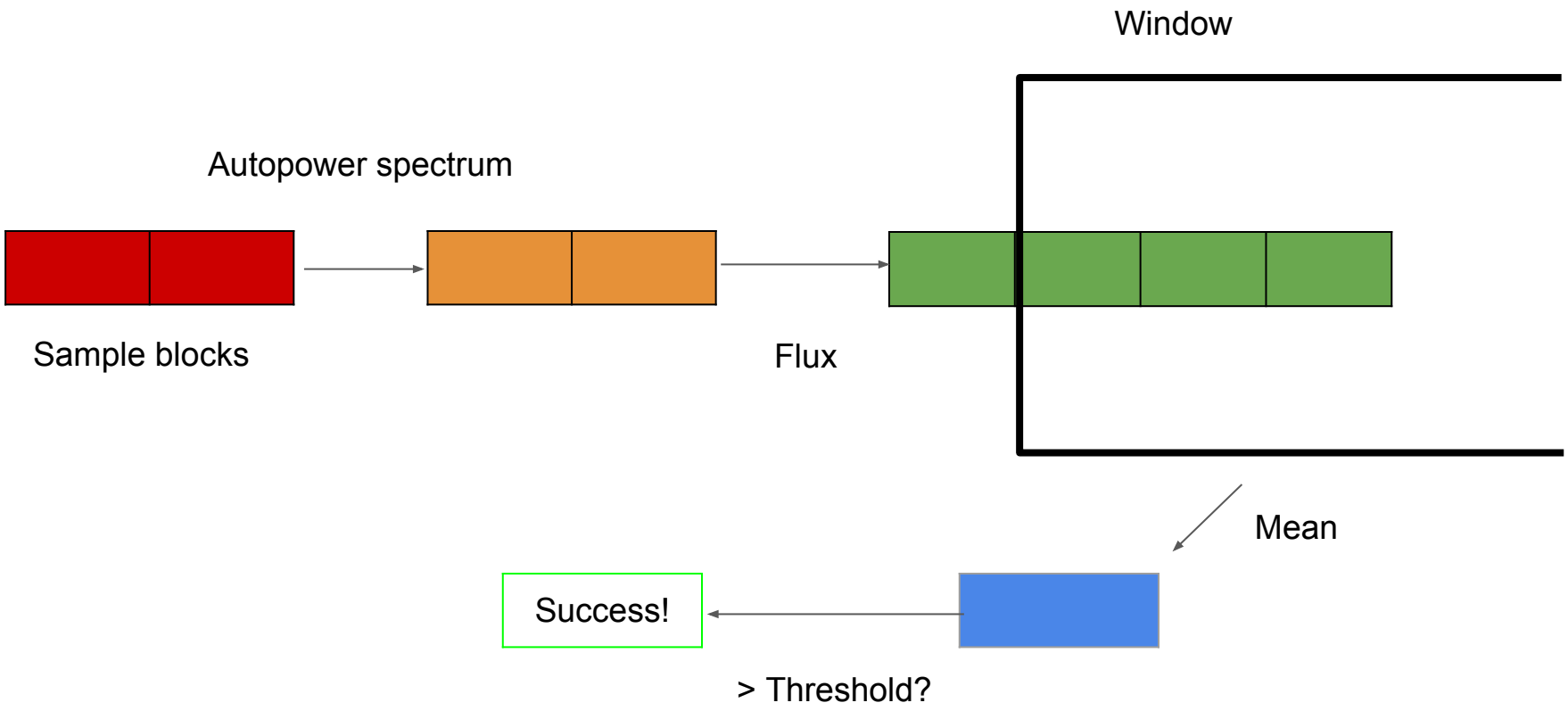
```
(defmethod plot-wave ((wave <static-wave>))
  (let ((series (audio-series wave))
        (filename (in-system-path "test" "png")))
    (with-plots (*standard-output* :debug nil)
      (gp-setup :output filename)
      (plot (lambda ()
              (loop for p in series
                    do (format t "~&~{~a~^ ~}" p)))
            :using '(1 2)
            :with '(lines notitle)))
    filename))
```

Analysis

Anna Wszeborowska - Processing music on the fly with Python

- https://www.youtube.com/watch?v=at2NppqIZok
- https://github.com/aniawsz/rtmonoaudio2midi

Algorithm


Window
Autopower spectrum
Sample blocks
Flux
Mean
Success!
> Threshold?

Autopower spectrum

```
(defun autopower-spectrum (analyzer data)
  (let* ((window-size (length data))
         (windowed (map 'vector #'* data (hwindow analyzer)))
         (padded (concatenate 'vector (inner-pad analyzer) windowed))
         (spectrum (map 'vector #'(lambda (x) (/ x window-size))
                        (bordeaux-fft:sfft padded)))
         (autopower (map 'vector #'(lambda (x y) (abs (* x y)))
                         spectrum
                         (map 'vector #'conjugate spectrum))))
    (slice autopower (cons 0 window-size))))
```

Flux

```
(defun diff (a b) (max (- a b) 0))

(let ((flux (reduce '+ (map 'vector #'diff
                            spectrum
                            (last-spectrum analyzer)))))
```

Framework

```
(defgeneric analyze-buffer (processor buffer))

(defgeneric setup-processor (processor wave))

(defgeneric cleanup-processor (processor))

(defun load-wave (source)
  (make-instance '<static-wave>
                 :audio-data (read-audio-data source)))

(defun from-mic ()
  (make-instance '<real-time-wave>))
```

Usage

```
(defun play-test-audio ()
  (let ((wave (load-wave (test-file-path))))
    (analyze-wave wave
                  '<player>
                  '<spectrum-analyzer>)))

(defun analyze-data-from-mic ()
  (let ((wave (from-mic)))
    (analyze-wave wave '<spectrum-analyzer>)))
```

Usage

```
(defun check-buffer-for-spike (buffer analyzer)
  (when (contains-peak-p analyzer (autopower-spectrum analyzer
                                                  (audio-data buffer)))
    (format t "peak! frame ~s, time ~s ~%"
            (frame-index buffer) (ts buffer))))

(defmethod analyze-buffer ((analyzer <spectrum-analyzer>) buffer)
  (check-buffer-for-spike buffer analyzer))
```

Usage

```
(defmethod setup-processor ((player <player>) wave)
  (initialize)
  (setf (astream player) (open-default-stream … ))
  (start-stream (astream player)))

(defmethod analyze-buffer ((player <player>) buffer)
  (write-stream (astream player) (audio-data buffer)))


(defmethod cleanup-processor ((player <player>))
  (portaudio::stop-stream (astream player))
  (close-stream (astream player))
  (terminate))
```

```
(defun analyze-wave (wave &rest processor-classes)
  (let ((processors (mapcar #'make-instance processor-classes)))
    (dolist (p processors)
      (setup-processor p wave))
    (for-each-buffer wave #'(lambda (buffer)
                              (dolist (p processors)
                                (analyze-buffer p buffer))))
    (dolist (p processors)
      (cleanup-processor p))))
```

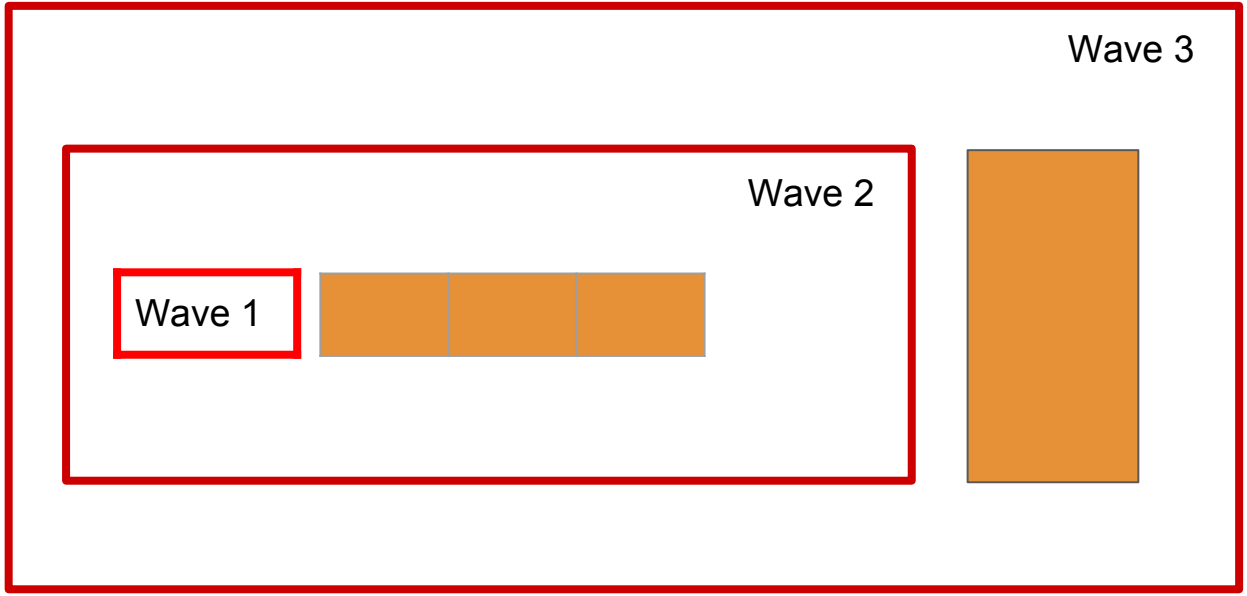
```
(defun analyze-wave (wave &rest processor-classes)
  (let ((processors (mapcar #'make-instance processor-classes)))
    (dolist (p processors)
      (setup-processor p wave))
    (for-each-buffer wave #'(lambda (buffer)
                              (dolist (p processors)
                                (analyze-buffer p buffer))))
    (dolist (p processors)
      (cleanup-processor p))))
```

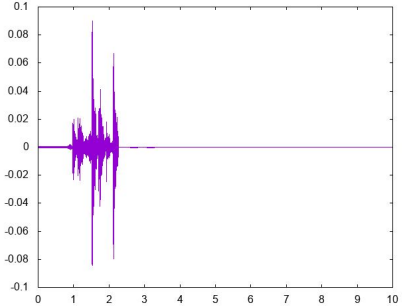
Magic!

```
(defmethod for-each-buffer ((wave <real-time-wave>) func)
  (let* ((buffer (make-buffer wave))
         (buffer-size (frame-count buffer))
         (idx 0))
    (with-audio
      (with-default-audio-stream (astream ...)
        (loop while t
              do (fill-buffer (audio-data buffer)
                              (read-stream astream) 0 buffer-size)
            (setf (frame-index buffer) idx)
            (setf (ts buffer) (* (num-channels buffer) (/ idx (sample-rate buffer))))
            (funcall func buffer)
            (incf idx buffer-size))))))
```

Bonus!

Filters


Wave 3
Wave 2
Wave 1


0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
0
1
2
3
4
5
6
7
8
9
10

Buffer in = Buffer out

```
(defgeneric filter-buffer (filter buffer))
(defgeneric setup-filter (filter wave))
(defgeneric cleanup-filter (filter))

(defun filter-wave (wave &rest filter-classes)
  (make-instance '<filtered-wave>
                 :parent-wave wave
                 :filters filter-classes))
```

```
(defun plot-test-amplification (&optional (ratio 2))
  (let ((wave (load-wave (test-file-path))))
    (plot-wave (filter-wave wave (list '<amplifier> :ratio ratio)))))
```

```
(defmethod for-each-buffer ((wave <filtered-wave>) func)
  (let ((filters-objects (mapcar #'(lambda (x) (if (listp x)
                                                 (apply 'make-instance x)
                                                 (make-instance x))) (filters wave))))
    (dolist (p filters-objects)
      (setup-filter p wave))
    (for-each-buffer (parent-wave wave)
                     #'(lambda (buffer)
                         (let ((new-data (reduce #'(lambda (b filter)
                                                     (filter-buffer filter b))
                                                 filters-objects
                                                 :initial-value (audio-data buffer))))
                           (setf (audio-data buffer) new-data)
                           (funcall func buffer))))
    (dolist (p filters-objects)
      (cleanup-filter p))))
```

```
(defmethod audio-data ((wave <filtered-wave>))
  (let ((data (make-array (frame-count (parent-wave wave))
                          :element-type 'single-float
                          :initial-element 0.0))
        (idx 0))
    (for-each-buffer wave #'(lambda (buffer)
                              (fill-buffer data (audio-data buffer)
                                           0 (frame-count buffer) idx)
                              (incf idx (frame-count buffer))))
    ;; (audio-data (parent-wave wave))))
    data))
```

TODO

- Record for later analysis
- Saving sound
- Multichannel processing
- Wave mixing
- Support multiple sound formats
- Slicing/cutting/stretching
- Proper matrix operations

Credits

- cl-portaudio
- cl-wav
- cl-arrows
- eazy-gnuplot
- bordeaux-fft
- cl-slice

0 << Common Lisp << Python

Lomer's

Opposite Railway Station.

805 GEORGE STREET,
SYDNEY.

Demo time!

Questions!