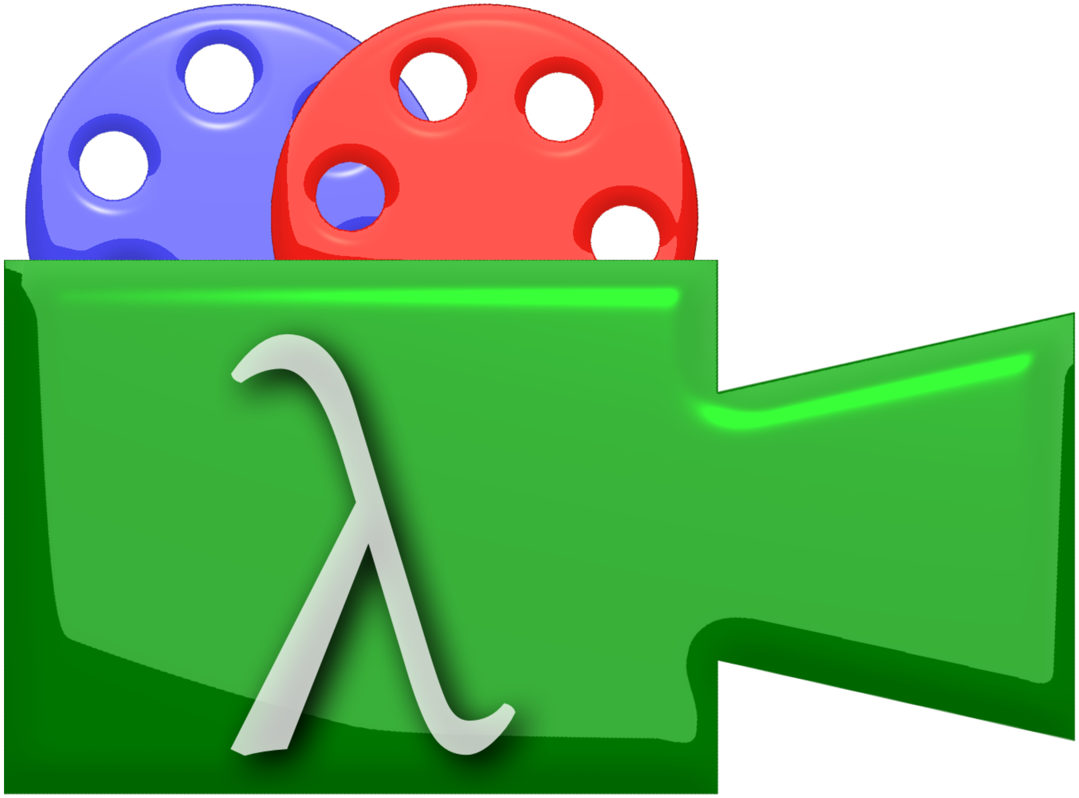
Movies as Programs

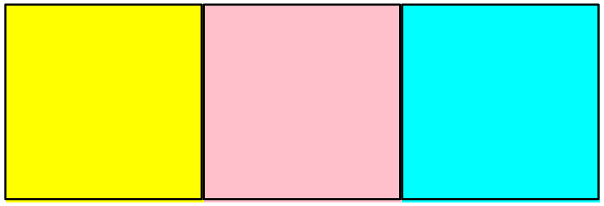


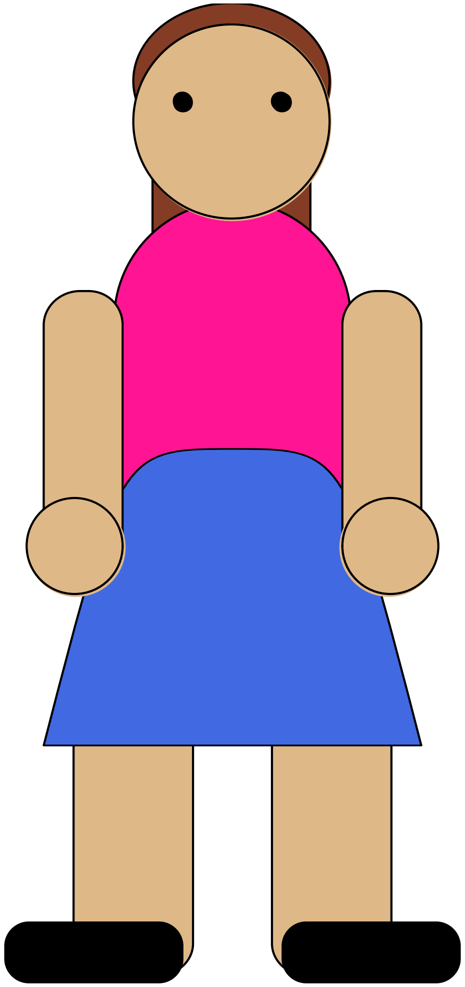
Leif Andersen

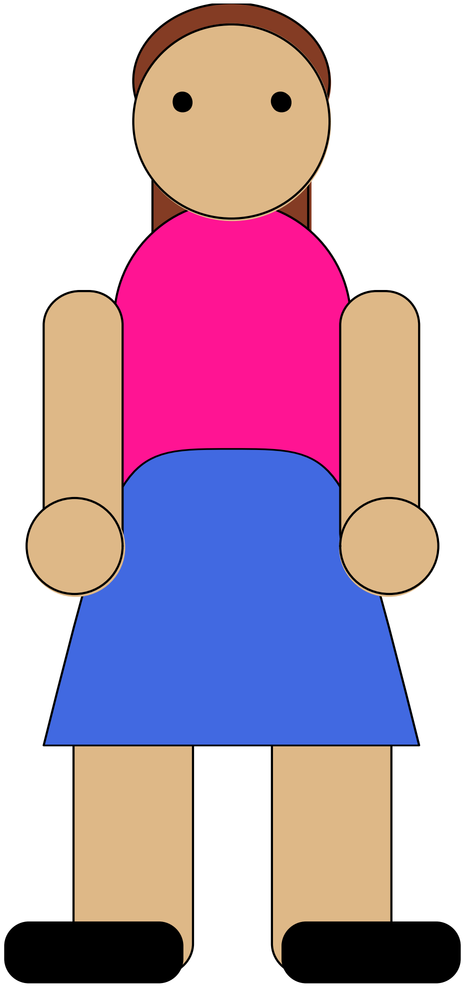
Accessibility

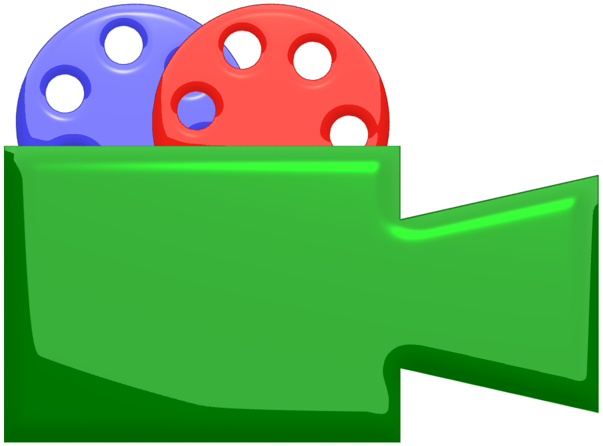
```
(prominent code)
```

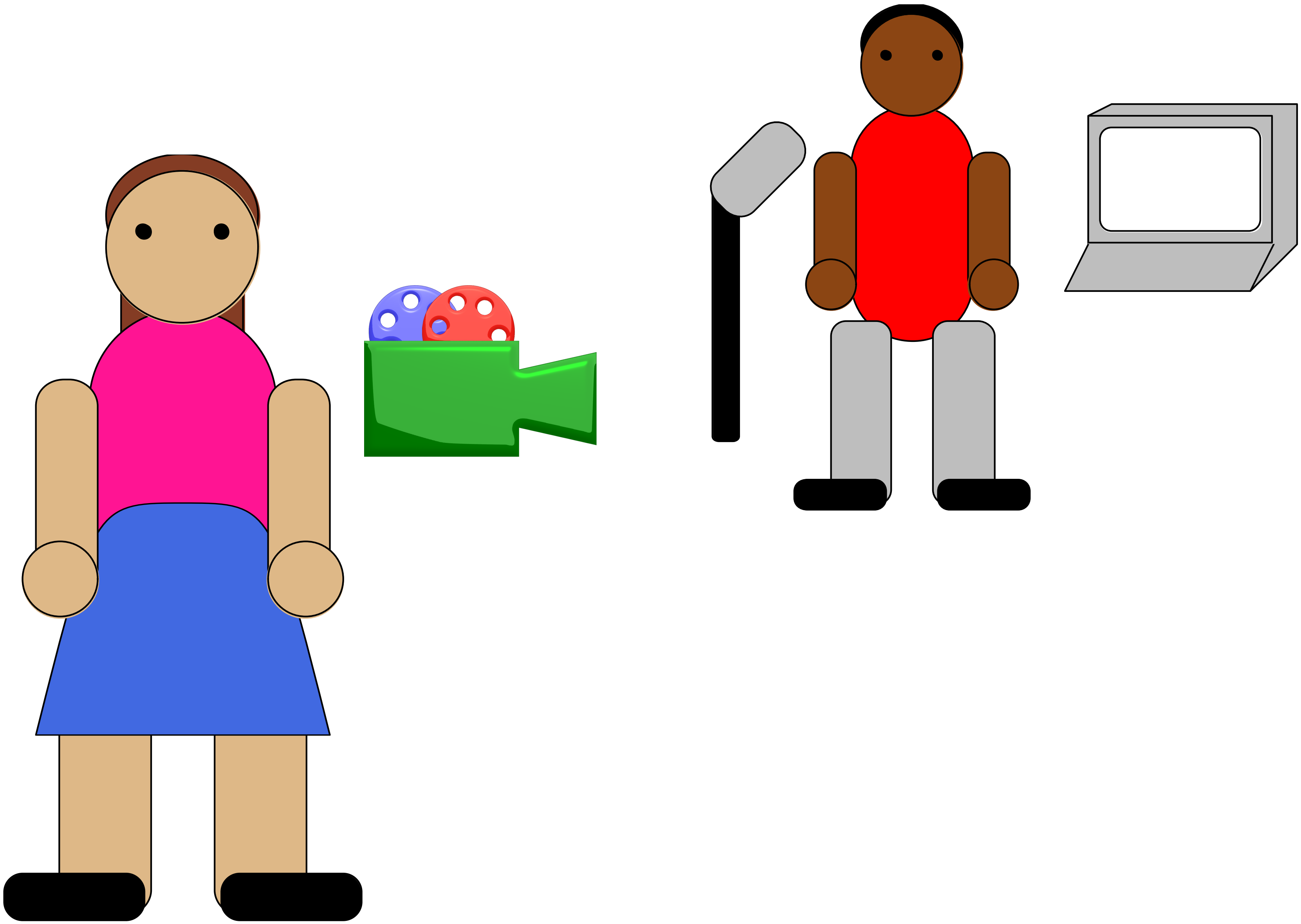
```
(some code)
```

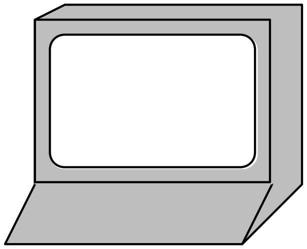


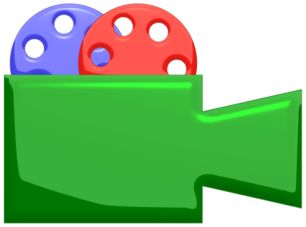


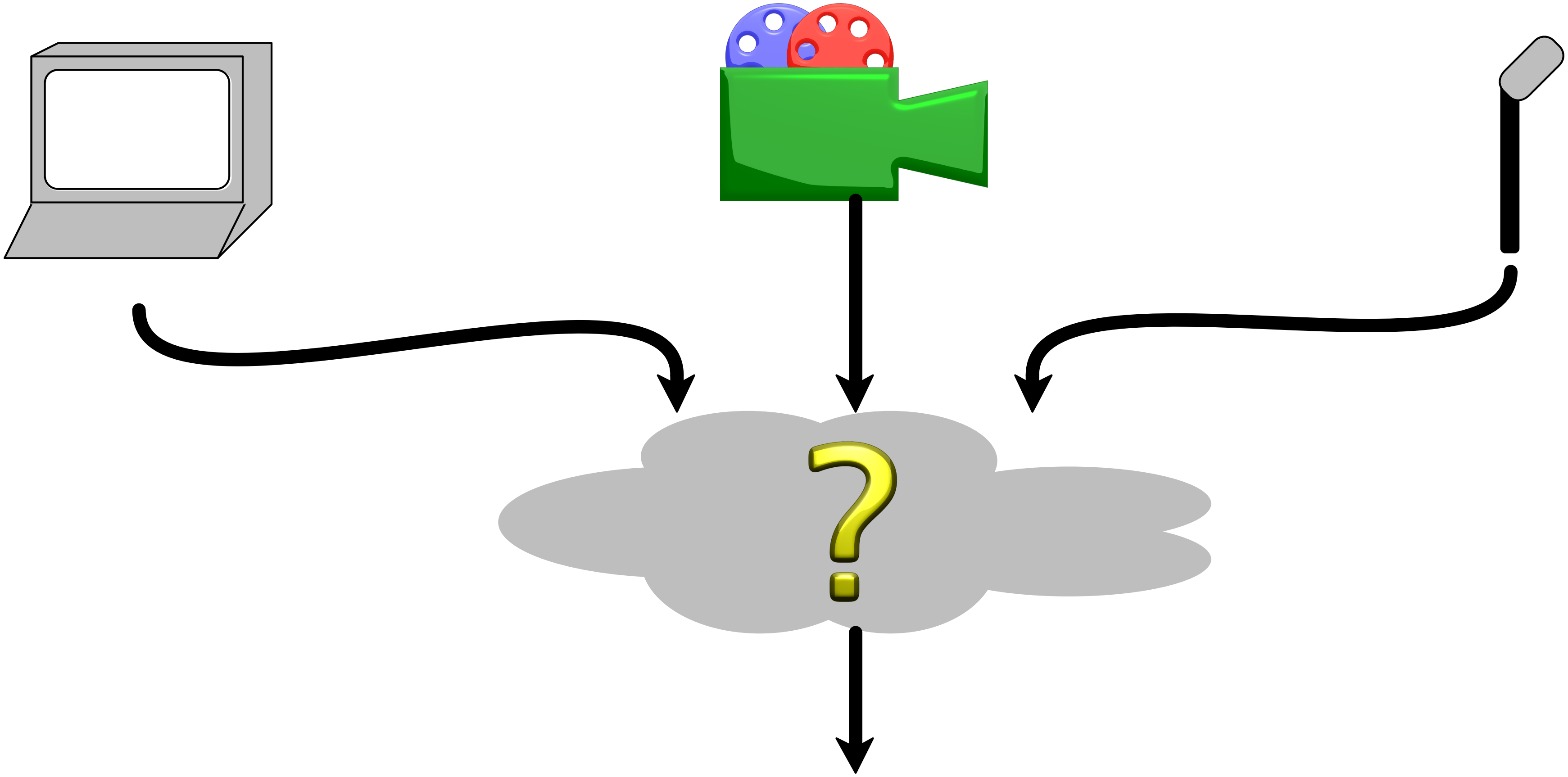


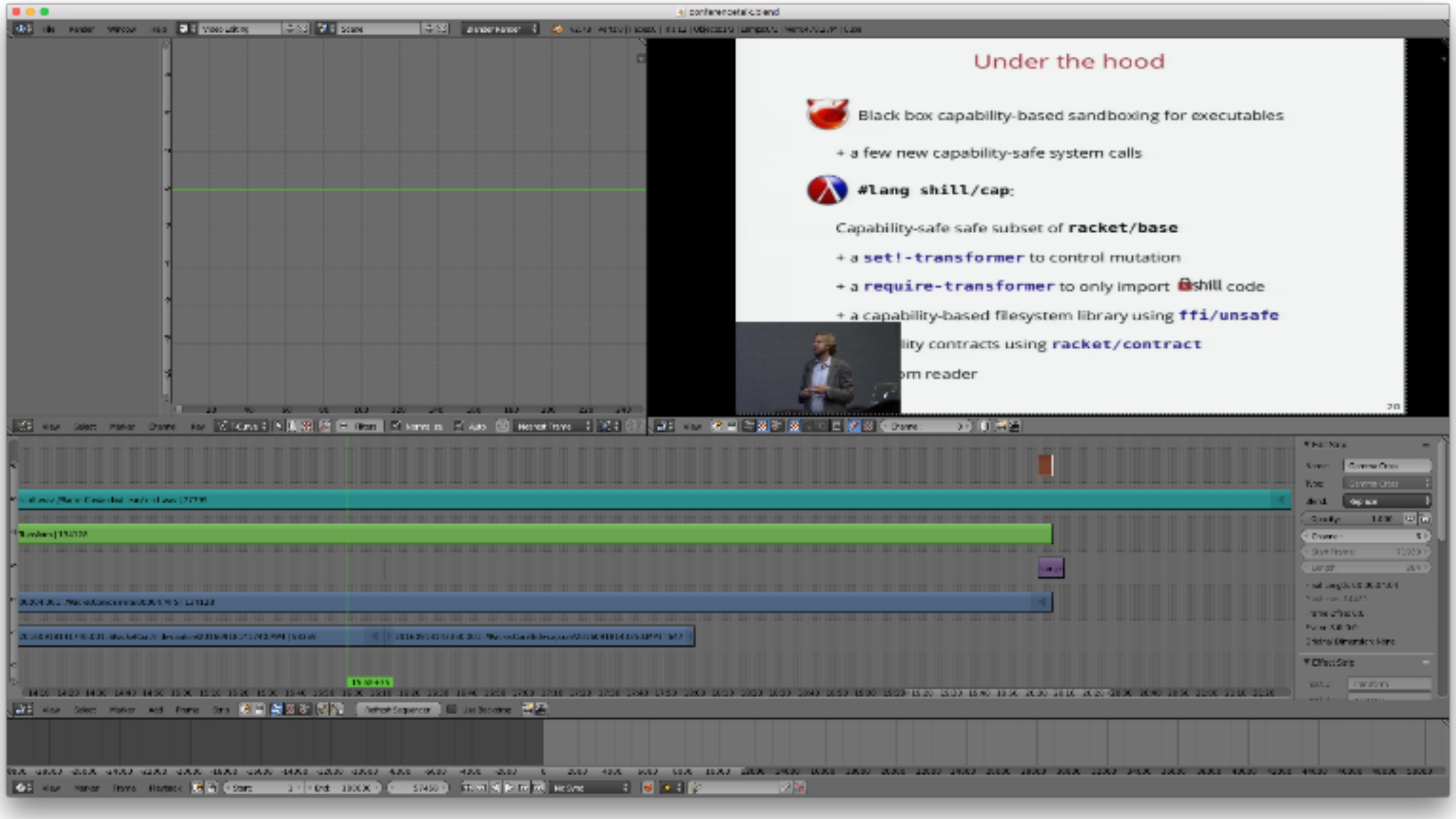












Under the hood

Black box capability-based sandboxing for executables

+ a few new capability-safe system calls

`#lang shill/cap`:

Capability-safe safe subset of **`racket/base`**

+ a `set!-transformer` to control mutation

+ a `require-transformer` to only import shill code

+ a capability-based filesystem library using `ffi/unsafe`

lity contracts using `racket/contract`

om reader

20

One down

One down

19 more to go…

We Need Automation

The Landscape

Tool	Example	Experience
Plugin-Ins	Blender Script, AE Script	
UI Automation (Macros)	Apple Script	
Shell Scripts	FFmpeg, AVISynth	

The Landscape

Tool	Example	Experience
Plugin-Ins	Blender Script, AE Script	🙁
UI Automation (Macros)	Apple Script	
Shell Scripts	FFmpeg, AVISynth	

The Landscape

Tool	Example	Experience
Plugin-Ins	Blender Script, AE Script	🙁
UI Automation (Macros)	Apple Script	😧
Shell Scripts	FFmpeg, AVISynth	

The Landscape

Tool	Example	Experience
Plugin-Ins	Blender Script, AE Script	🙁
UI Automation (Macros)	Apple Script	😟
Shell Scripts	FFmpeg, AVISynth	😬

We have a problem…

We have a problem...

We want to solve it in the problem domain's own language...

We have a problem...

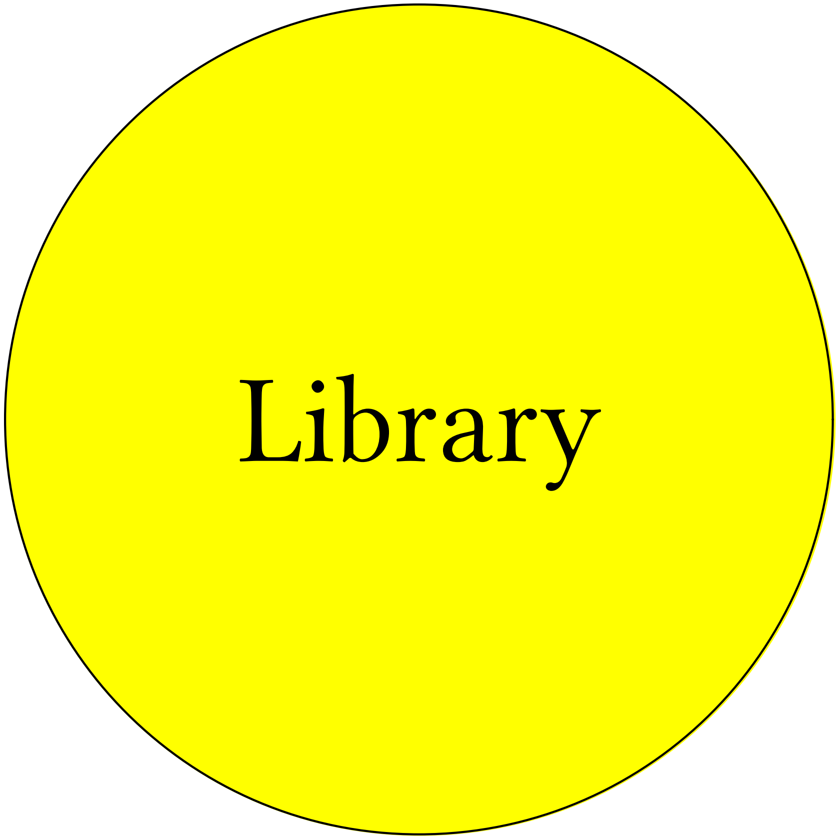
We want to solve it in the problem domain's own language...

DSLs are the

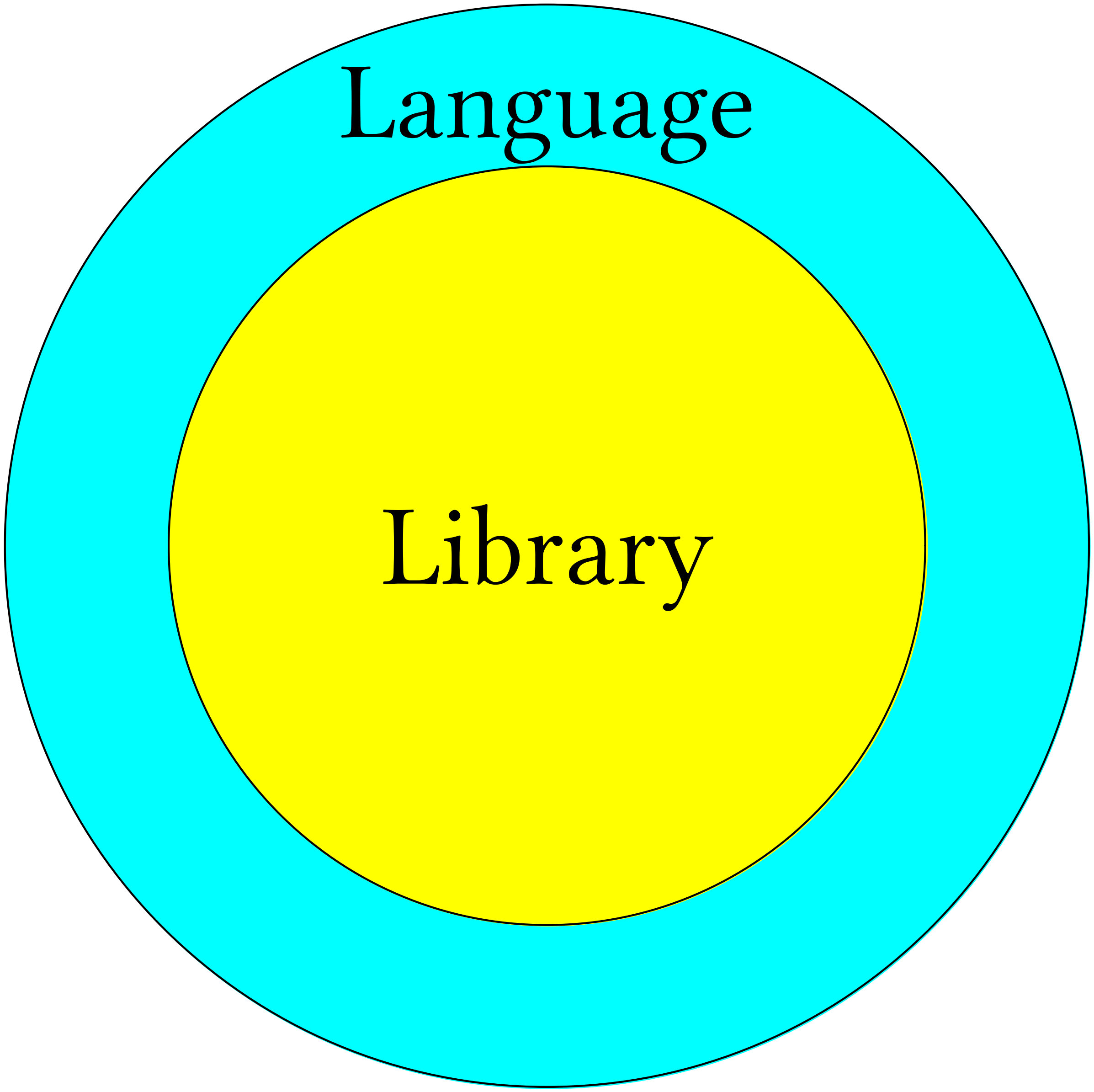
"Ultimate Abstraction"

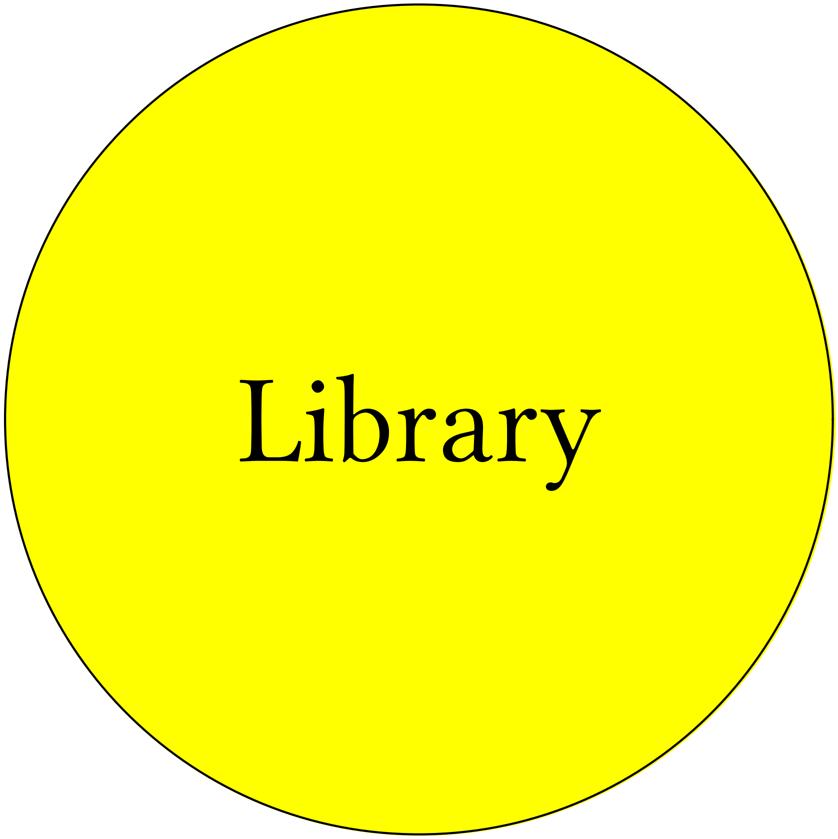
Paul Hudak

Make a DSL!

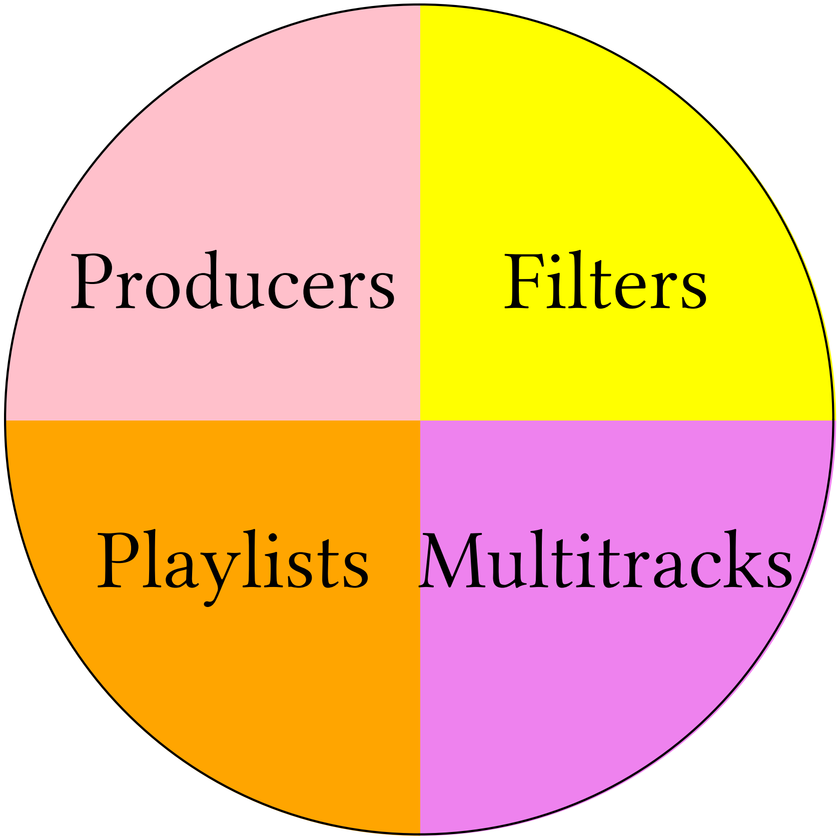


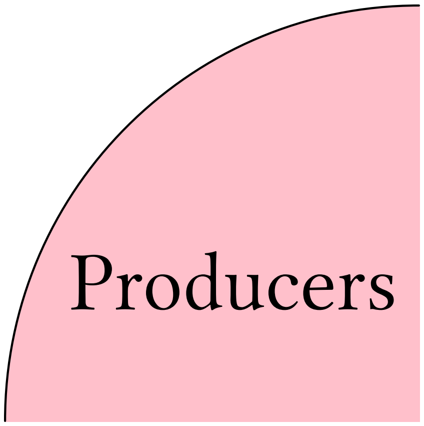
Library


Language
Library



Library


Producers
Filters
Playlists
Multitracks


Producers

Producers

Producers

`render : Producer` → 📄

Producers

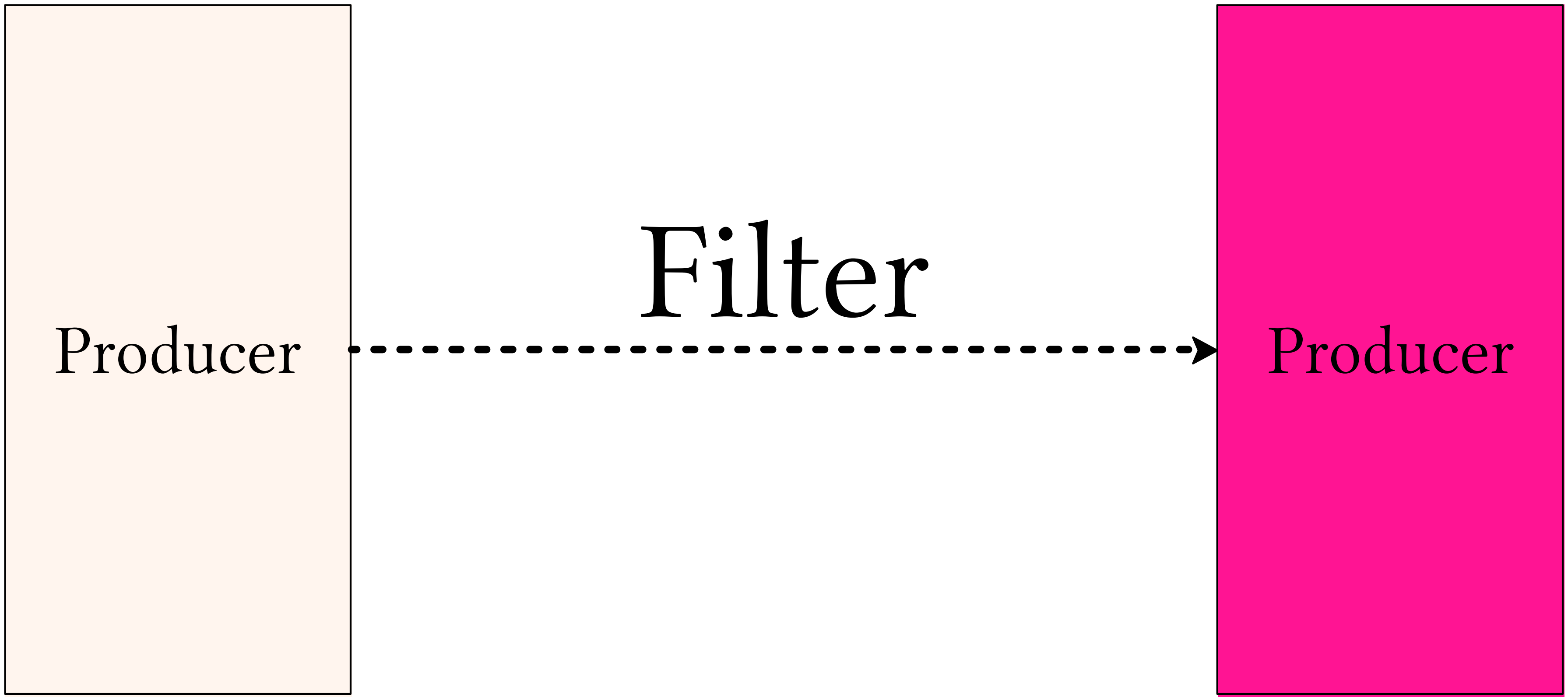
```
render : Producer → 📄
clip : String → Producer
```

Producers

```
render : Producer → 📄

clip : String → Producer
```

```
(render (clip "demo.mp4")) ⇒
```

Filters



Producer

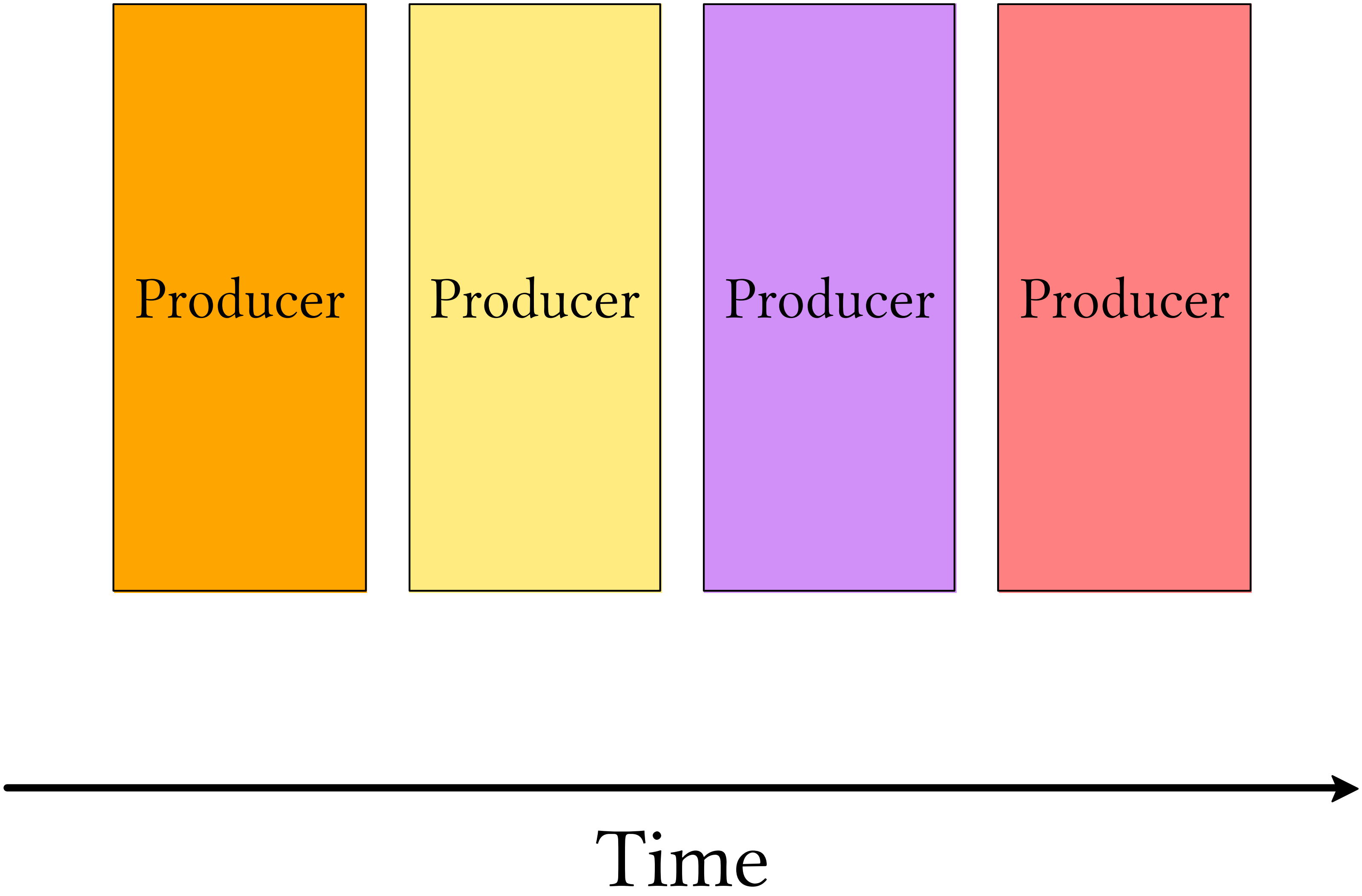
Filter

Producer

```
(attach-filter bunny-clip (sepia-filter))
```

```
(attach-filter bunny-clip (sepia-filter))
```

Playlists



Producer

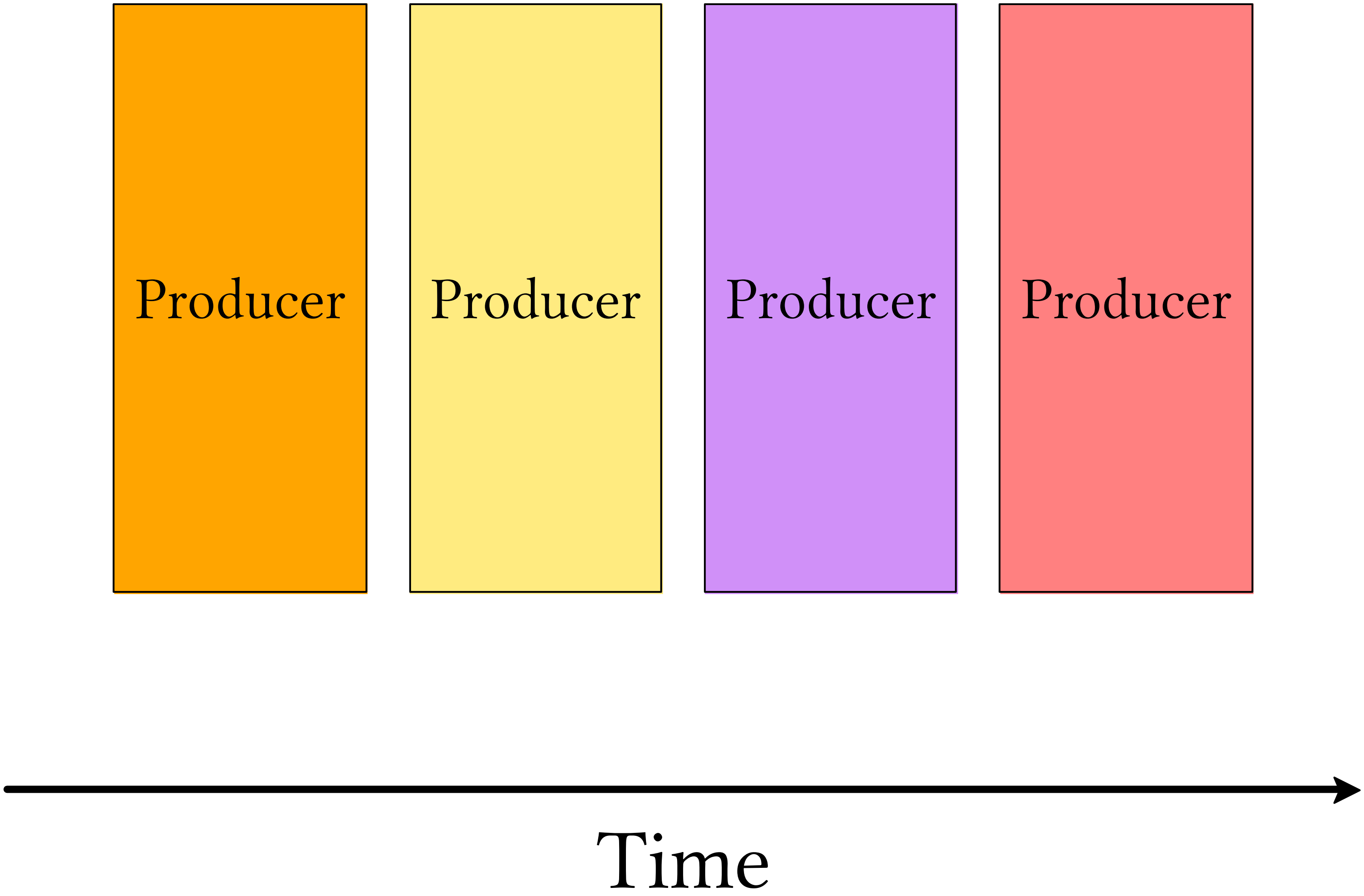
Producer

Producer

Producer

Time

```
(playlist (clip "jumping.mp4")
          (clip "flying.mp4"))
```



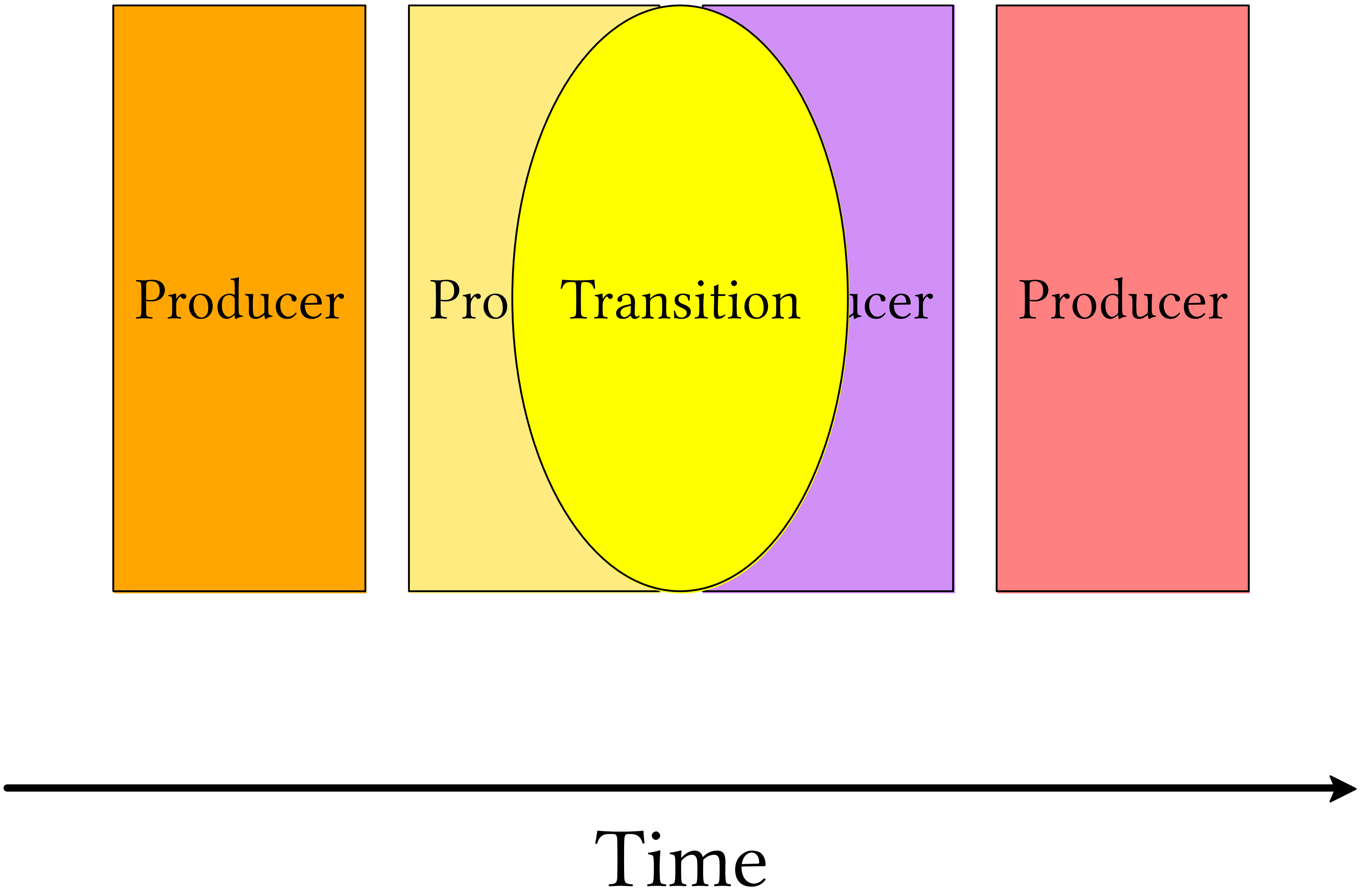
Producer

Producer

Producer

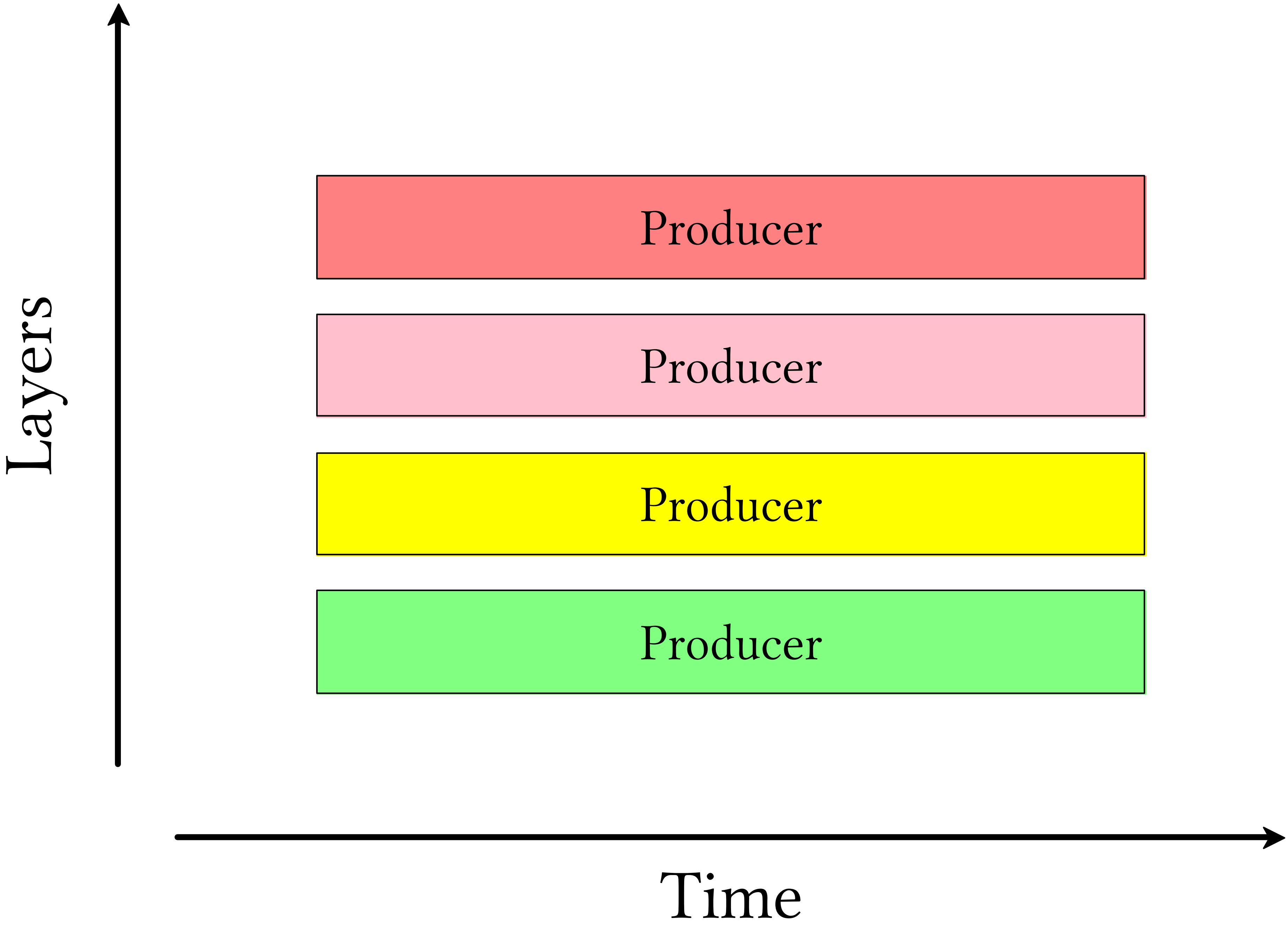
Producer

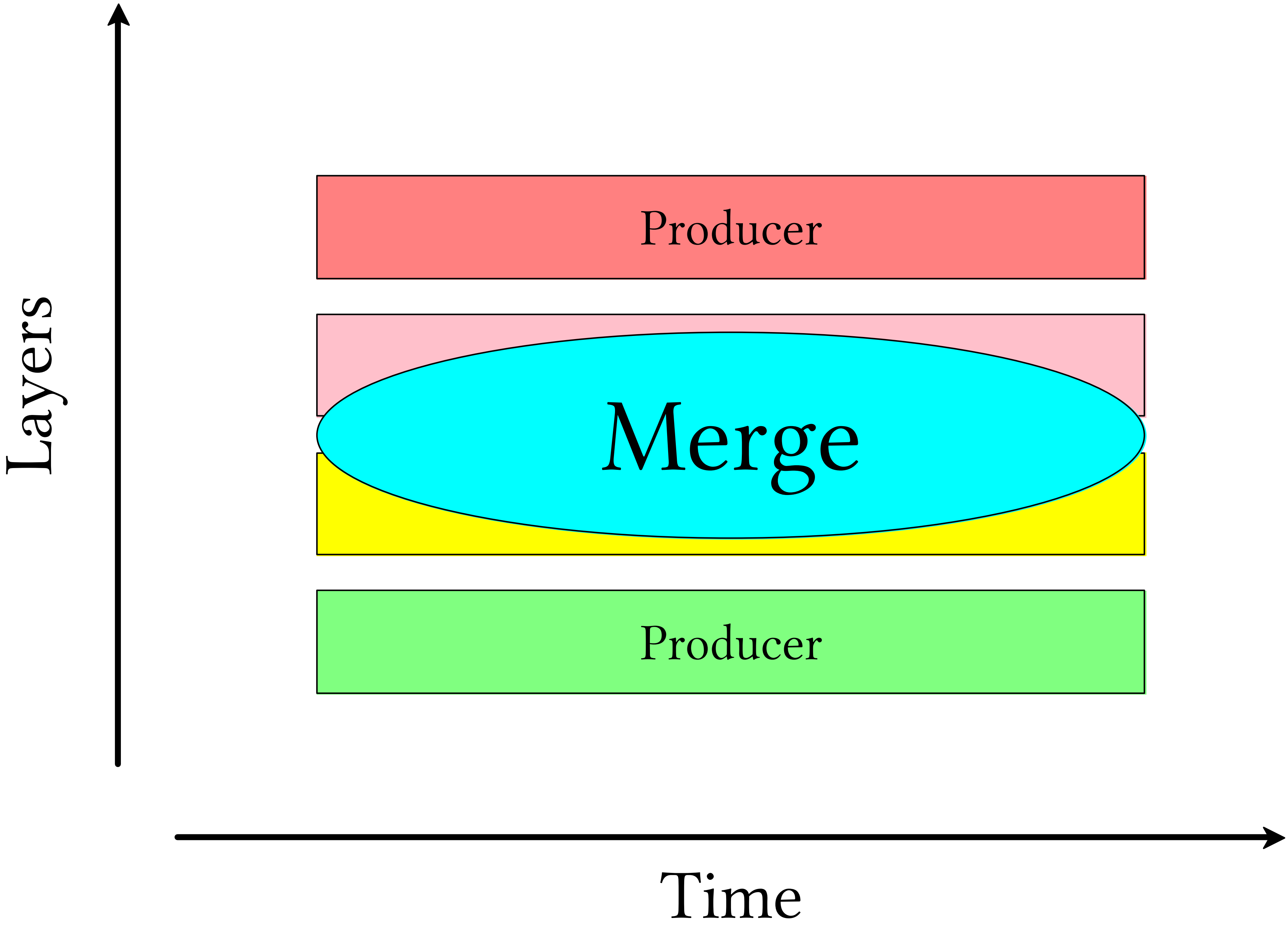
Time


Producer
Pro
Transition
ucer
Producer
Time

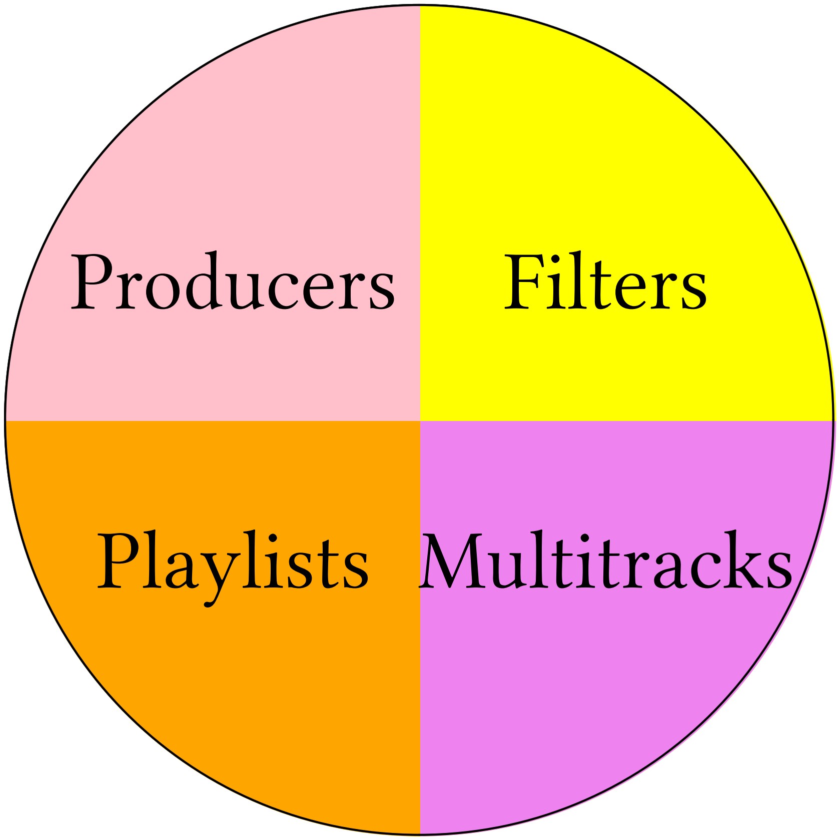
```
(playlist (clip "jumping.mp4")
          (fade-transition 1)
          (clip "flying.mp4"))
```

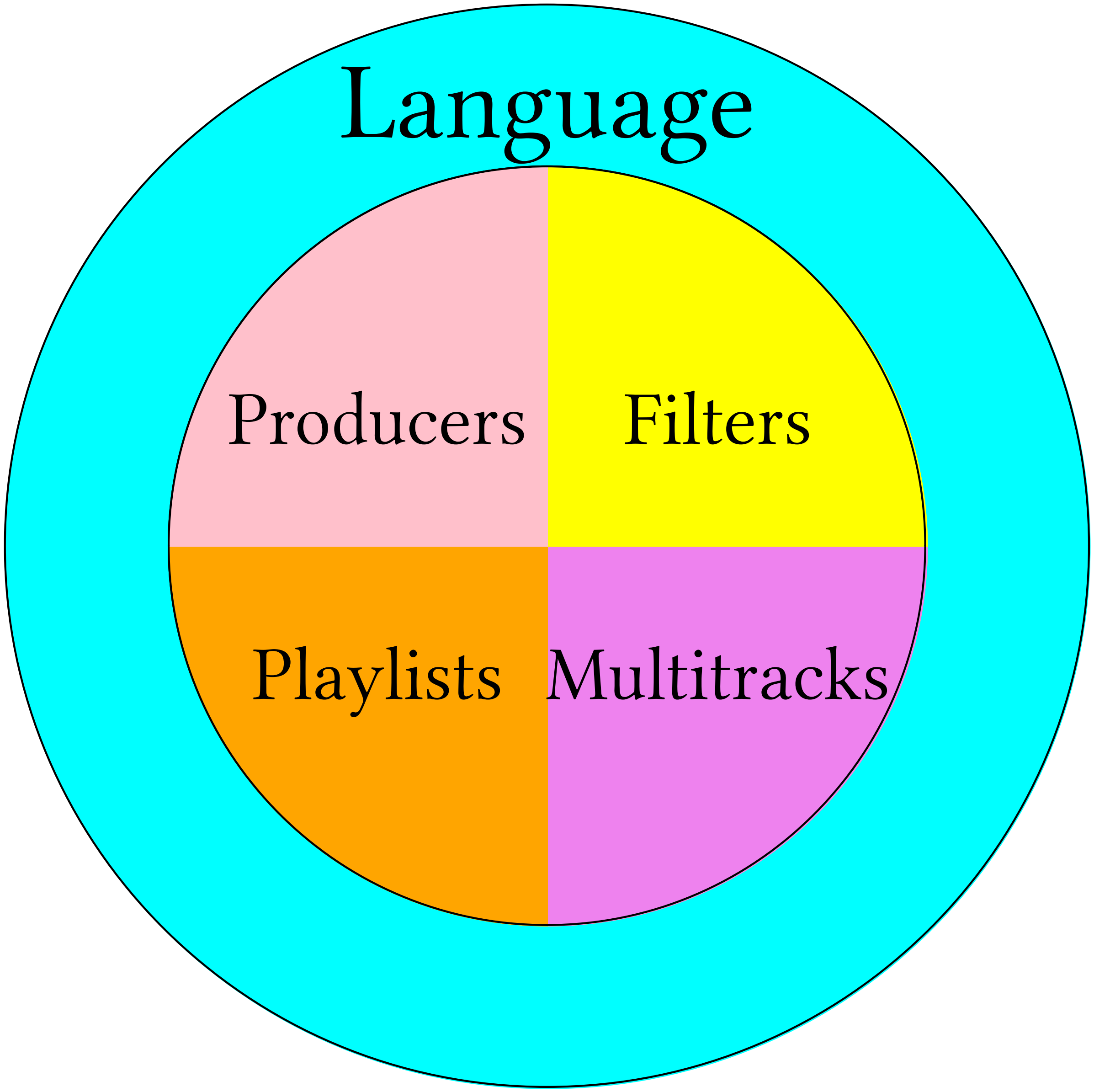
Multitracks


Layers
Producer
Producer
Producer
Producer
Time


Layers
Producer
Merge
Producer
Time

```
(define WIDTH 1920)
(define HEIGHT 1080)
(multitrack (color "black")
            (overlay-merge 0 0 (/ WIDTH 2) HEIGHT)
            (clip "running.mp4")
            (overlay-merge (/ WIDTH 2) 0 (/ WIDTH 2) HEIGHT)
            (clip "flying.mp4"))
```


Producers
Filters
Playlists
Multitracks


Language
Producers
Filters
Playlists
Multitracks

mosaic.vid

```
#lang video
;; Create a mosaic of four videos
(for/vertical ([i (in-range 2)])
  (for/horizontal ([j (in-range 2)])
    (external-video "branded.vid"
      (clip "logo.png")
      (clip (format "~aX~a.mp4" i j)))))
```

mosaic.vid

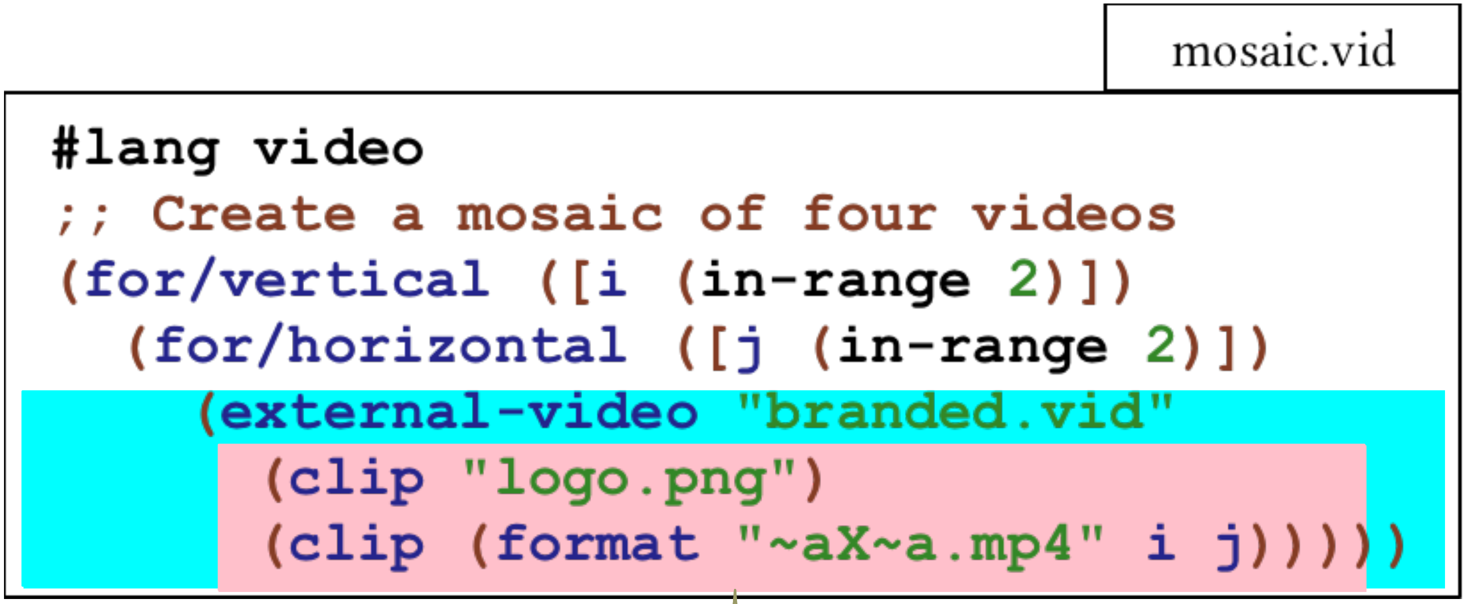
```
#lang video
;; Create a mosaic of four videos
(for/vertical ([i (in-range 2)])
  (for/horizontal ([j (in-range 2)])
    (external-video "branded.vid"
      (clip "logo.png")
      (clip (format "~aX~a.mp4" i j))))))
```

Primitives

mosaic.vid

```
#lang video
;; Create a mosaic of four videos
(for/vertical ([i (in-range 2)])
  (for/horizontal ([j (in-range 2)])
    (external-video "branded.vid"
      (clip "logo.png")
      (clip (format "~aX~a.mp4" i j)))))
```

List Comprehensions



mosaic.vid

```
#lang video
;; Create a mosaic of four videos
(for/vertical ([i (in-range 2)])
  (for/horizontal ([j (in-range 2)])
    (external-video "branded.vid"
      (clip "logo.png")
      (clip (format "~aX~a.mp4" i j)))))
```

Modules

mosaic.vid

```
#lang video
;; Create a mosaic of four videos
(for/vertical ([i (in-range 2)])
  (for/horizontal ([j (in-range 2)])
    (external-video "branded.vid"
     (clip "logo.png")
     (clip (format "~aX~a.mp4" i j))))))
```

branded.vid

```
#lang video/lib
;; Generate a branded video
(define-video (branded logo vid)
  logo
  (fade-transition 1)
  (multitrack logo
              (overlay 0 0 100 100)
              vid))
```

mosaic.vid

```
#lang video
;; Create a mosaic of four videos
(for/vertical ([i (in-range 2)])
  (for/horizontal ([j (in-range 2)])
            ideo "branded.vid"
            go.png")
      (clip (  rmat "~aX~a.mp4" i j)))))
```

Functions

branded.vid

```
#lang video/lib
;; Generate a branded video
(define-video (branded logo vid)
  logo
  (fade-transition 1)
  (multitrack logo
              (overlay 0 0 100 100)
              vid))
```

mosaic.vid

```
#lang video
;; Create a mosaic of four videos
(for/vertical ([i (in-range 2)])
  (for/horizontal ([j (in-range 2)])
    (external-video "branded.vid"
      (clip "logo.png")
      (clip (format "~aX~a.mp4" i j)))))
```

branded.vid

```
#lang video/lib
;; Generate a branded video
(define-video (branded logo vid)
  logo
  (fade-transition 1)
  (multitrack logo
              (overlay 0 0 100 100)
              vid))
```

```
#lang video
;; Create a mosaic of four videos
(for/vertical ([i (in-range 2)])
  (for/horizontal ([j (in-range 2)])
    (external-video "branded.vid"
      (clip "logo.png")
      (clip (format "~aX~a.mp4" i j)))))
```

```
#lang video
(clip "dragon.mp4")
;; Create a mosaic of four videos
(for/vertical ([i (in-range 2)])
  (for/horizontal ([j (in-range 2)])
    (external-video "branded.vid"
      (clip "logo.png")
      (clip (format "~aX~a.mp4" i j)))))
```

Implementing Video + Editing

Manual Editing

From Libraries to Languages

We make DSLs using

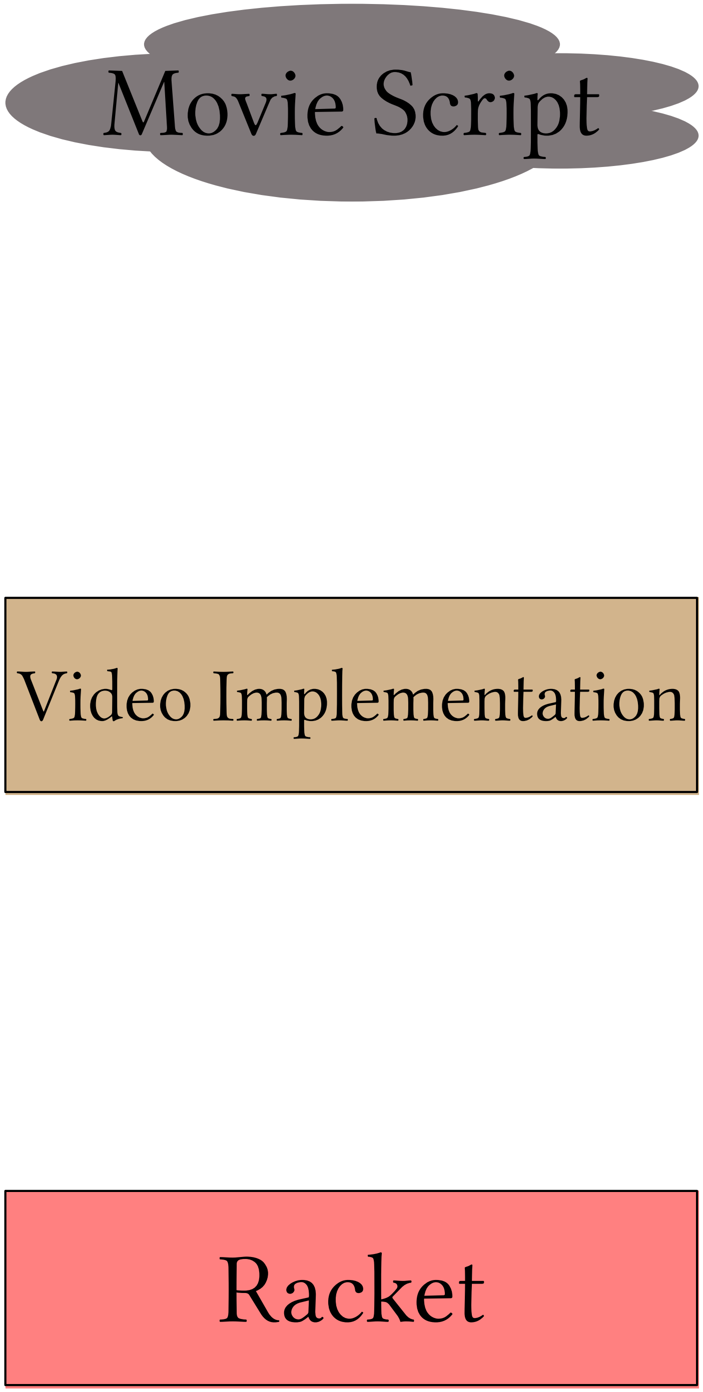
Linguistic Inheritance

We make DSLs using

Linguistic Inheritance

We make DSLs using

Linguistic Inheritance


Movie Script

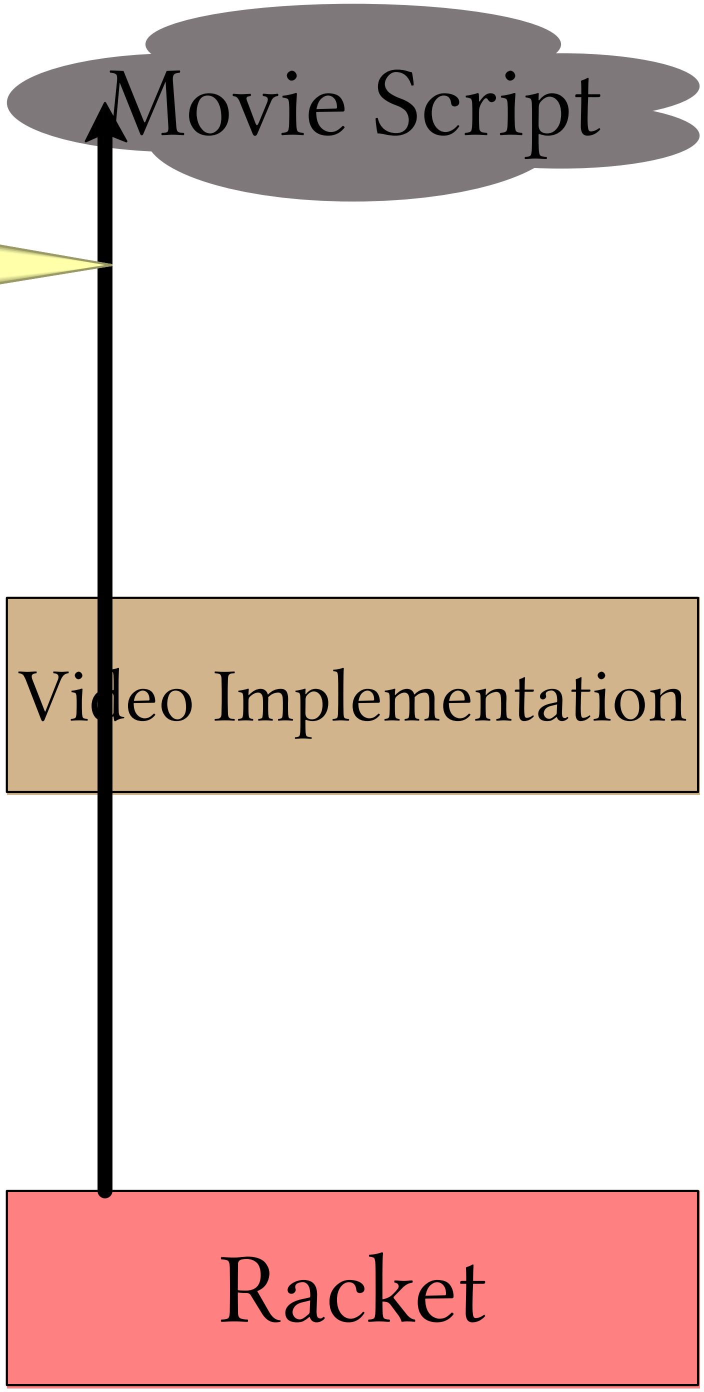
Video Implementation

Racket

We make DSLs using

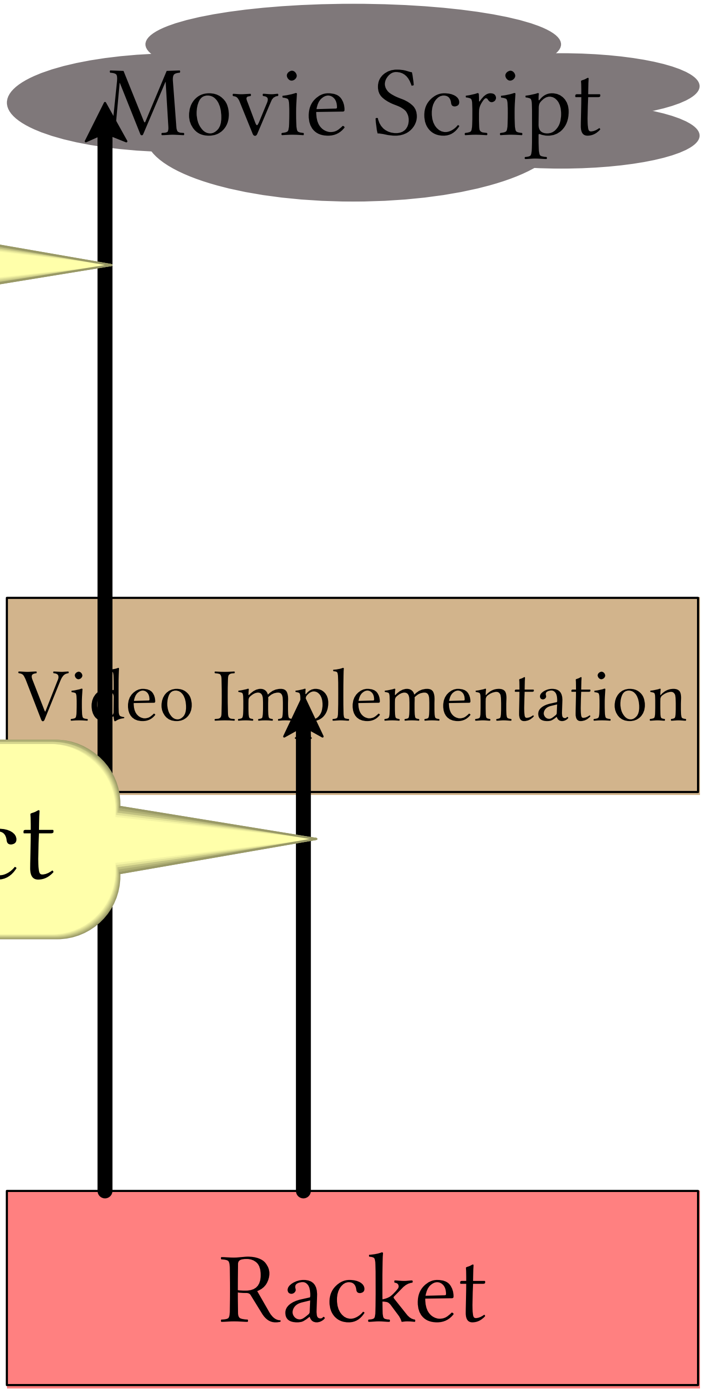
Linguistic Inheritance

Re-export construct



We make DSLs using

Linguistic Inheritance

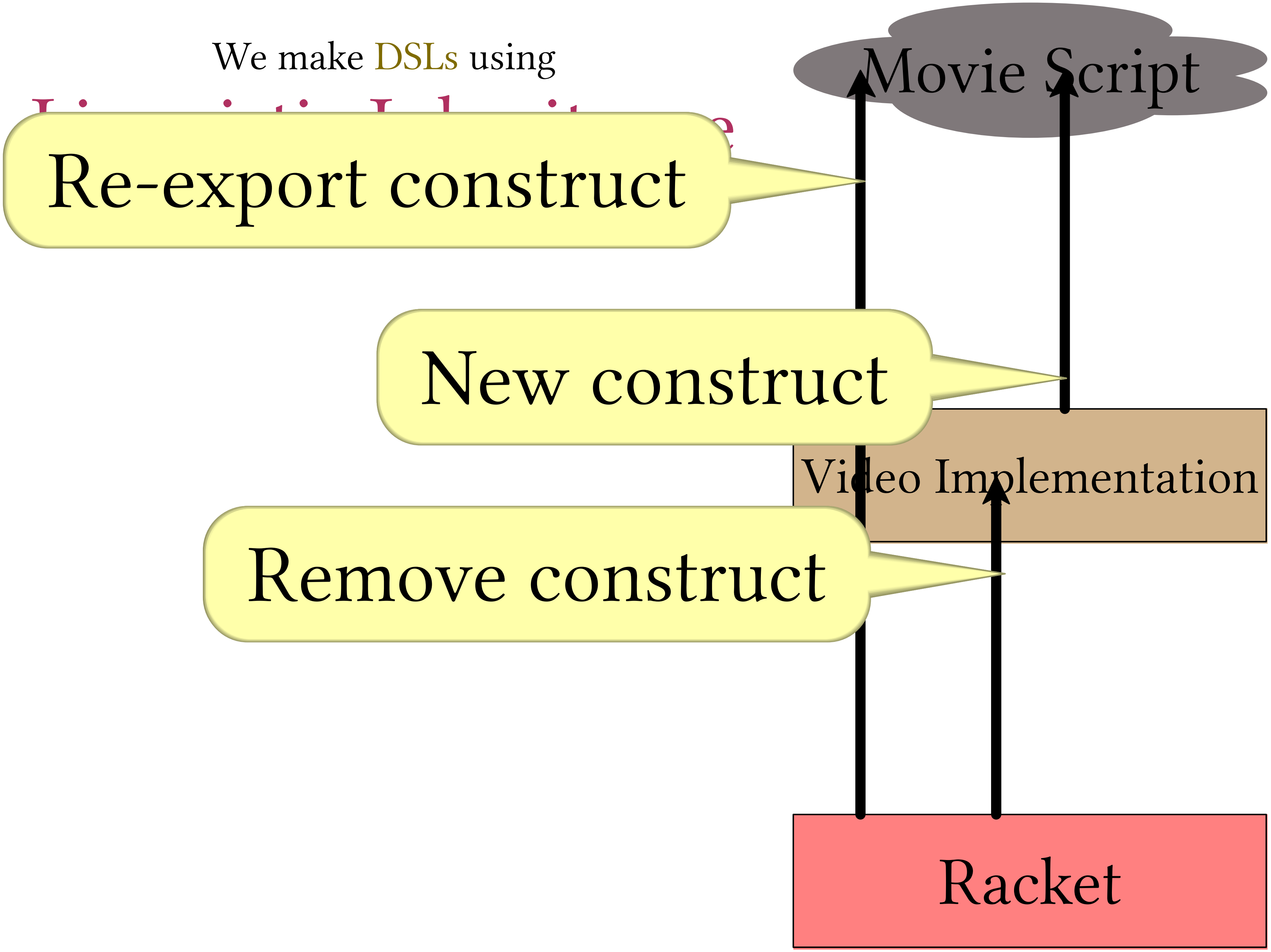


Re-export construct

Remove construct

We make DSLs using

Linguistic Inheritance



Re-export construct

New construct

Remove construct

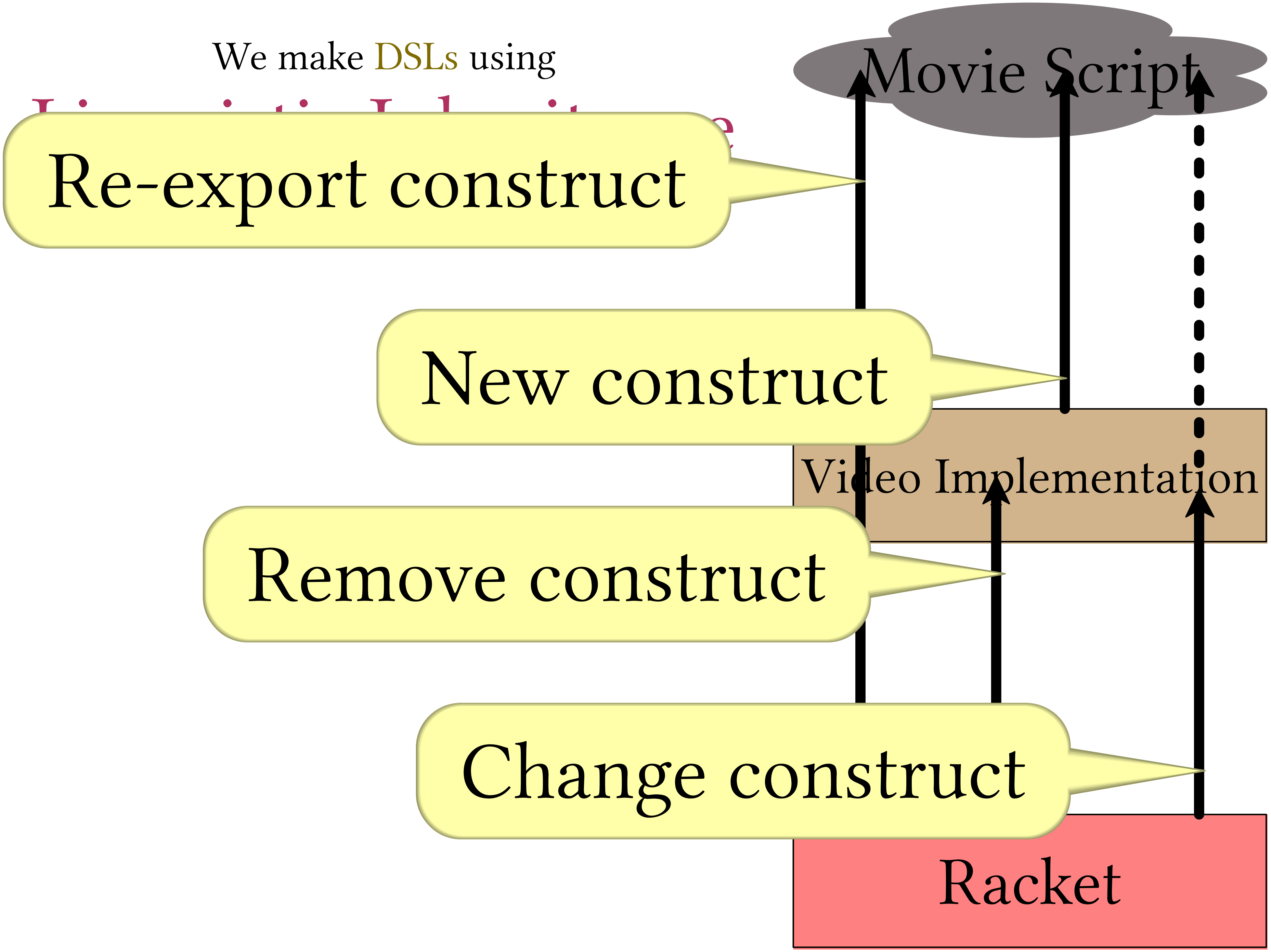
Movie Script

Video Implementation

Racket

We make DSLs using

Linguistic Inheritance



Movie Script

Re-export construct

New construct

Video Implementation

Remove construct

Change construct

Racket

```
(for/playlist ([scene (in-list scene-list)])
  (multitrack scene
              (overlay-merge 10 10 300 300)
              (clip "logo.mp4")))
```

```
(define (for/playlist seq body)
  (apply playlist
         (for/list ([i (in-list seq)])
           (body i))))
```

```
(define (for/playlist seq body)
  (apply playlist
         (for/list ([i (in-list seq)])
           (body i))))
```

```
> (for/playlist (list (clip "a.mp4")
                      (clip "b.mp4"))
    (λ (scene)
      (multitrack scene
                  (overlay-merge 10 10 300 300)
                  (clip "logo.mp4"))))
```

```
(define-macro (for/playlist seq . body)
  `(apply playlist
          (for/list ,seq
            ,@body)))
```

```
(for/playlist ([s (list (clip "a.mp4"))])
  (multitrack ...))
```

⟹ elaborates

```
(apply playlist
       (for/list ([s (list (clip "a.mp4"))])
         (multitrack ....)))
```

```
(for/playlist ([s (list (clip "a.mp4"))])
  (multitrack ...))
```

⟹ elaborates

```
(apply playlist
       (for/list ([s (list (clip "a.mp4"))])
         (multitrack ....)))
```

⟹ evaluates

```
#<playlist>
```

```
(let ([playlist 42])
  (for/playlist ....))
```

```
(let ([playlist 42])
  (for/playlist ....))
```

⟹ elaborates

```
(let ([playlist 42])
  (apply playlist ....))
```

```
(let ([playlist 42])
  (for/playlist ....))
```

⟹ elaborates

```
(let ([playlist 42])
  (apply playlist ....))
```

⟹ evaluates

```
(define-macro (for/playlist seq . body)
  `(apply playlist
          (for/list ,seq
            ,@body)))
```

```
> (let ([playlist 42])
    (for/playlist ([s (list (clip "a.mp4"))])
      (multitrack s
                  (overlay-merge 10 10 300 300)
                  (clip "logo.mp4"))))
```

```
(define-syntax-rule (for/playlist seq
                      body ...)
  (apply playlist
         (for/list seq
           body ...)))
```

```
(define-syntax-rule (for/playlist seq
                      body ...)
  (apply playlist
         (for/list seq
           body ...)))
```

```
> (let ([playlist 42])
    (for/playlist ([s (list (clip "a.mp4"))])
      (multitrack s
                  (overlay-merge 10 10 300 300)
                  (clip "logo.mp4"))))
```

lang-extension.rkt

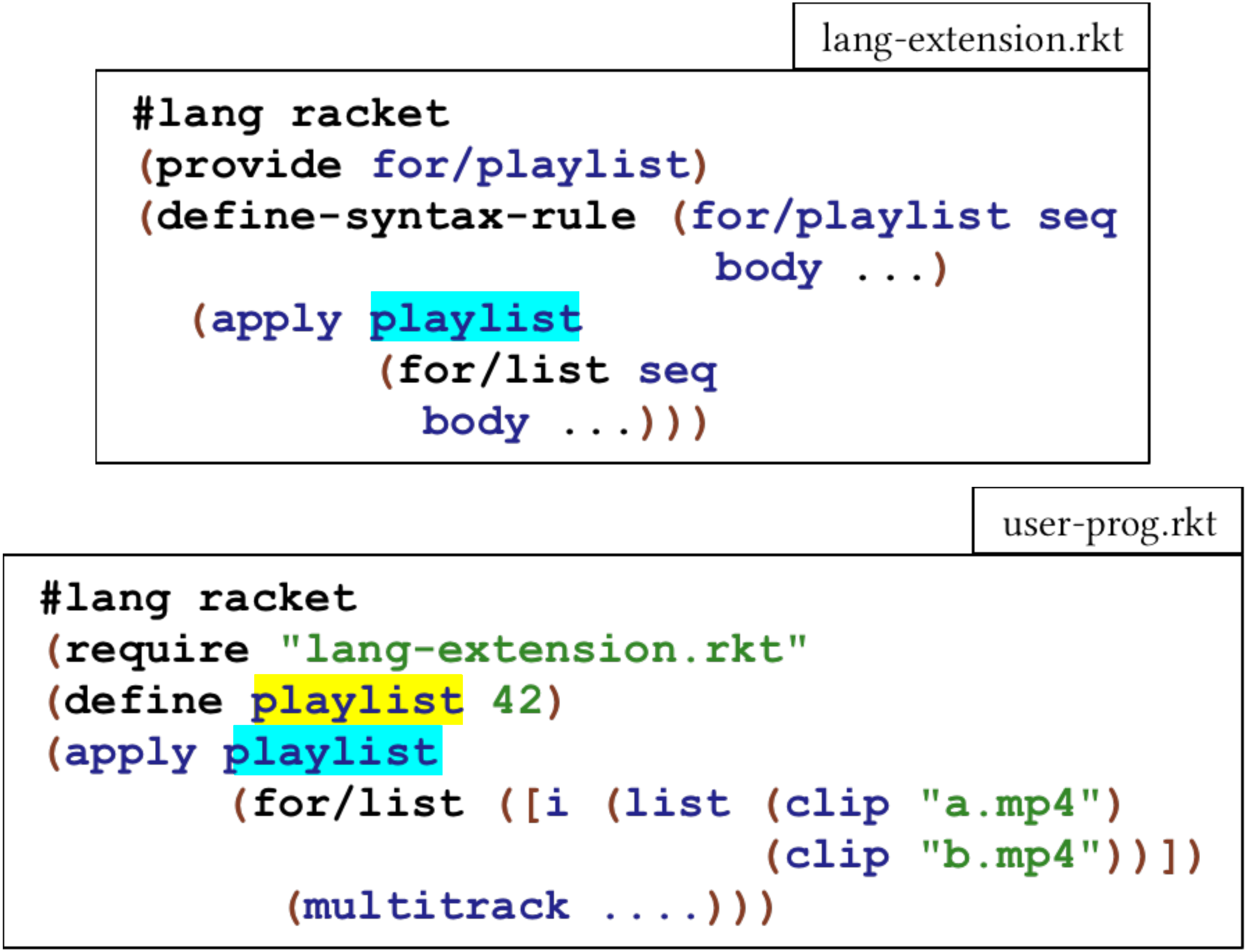
```
#lang racket
(provide for/playlist)
(define-syntax-rule (for/playlist seq
                      body ...)
  (apply playlist
         (for/list seq
           body ...)))
```

lang-extension.rkt

```
#lang racket
(provide for/playlist)
(define-syntax-rule (for/playlist seq
                      body ...)
  (apply playlist
         (for/list seq
           body ...)))
```

user-prog.rkt

```
#lang racket
(require "lang-extension.rkt"
(define playlist 42)
(for/playlist ([i (list (clip "a.mp4")
                        (clip "b.mp4"))])
  (multitrack ....))
```



Non-Local Language Features

```
#lang video

logo
(define logo ...)

talk
(define talk ...)

logo
```

```
#lang video

(provide vid)

(define logo ...)
(define talk ...)

(define vid (playlist
             logo
             talk
             logo))
```

Interposition Points

```
#%app
#%module-begin
```

```
(+ 1 2)
```

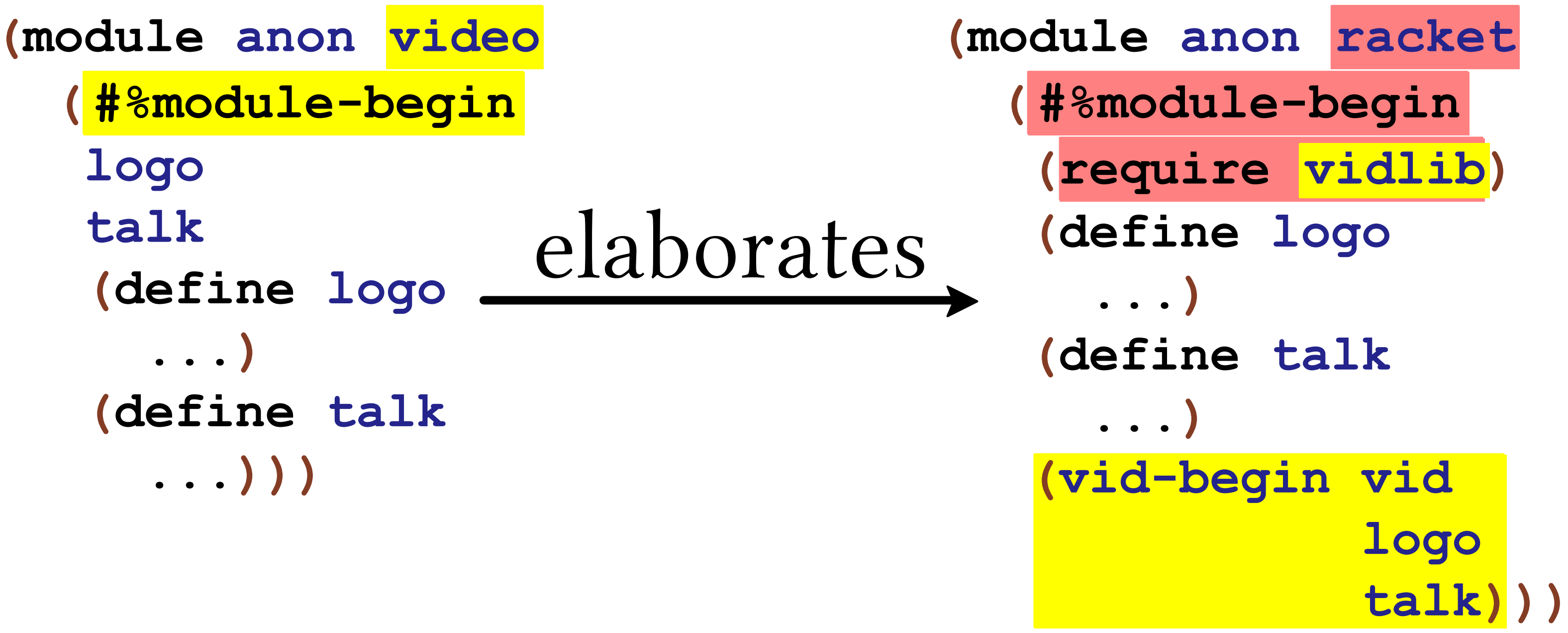
⟹ elaborates

```
(#%app + 1 2)
```

```
#lang video

logo
talk

;; Where
(define logo
...)
(define talk
...)
```

parses

```
(module anon video
  (#%module-begin
   logo
   talk
   (define logo
     ...)
   (define talk
     ...)))
```



```
(module anon video
  (#%module-begin
   logo
   talk
   (define logo
     ...)
   (define talk
     ...)))
```

elaborates

```
(module anon racket
  (#%module-begin
   (require vidlib)
   (define logo
     ...)
   (define talk
     ...)
   (vid-begin vid
              logo
              talk)))
```

```
#lang racket
(provide (rename-out [video-module-begin
                      #%module-begin]))
(define-syntax-rule (video-module-begin body ...)
    ... #%module-begin ...)
```

```
(require syntax/wrapping-modbeg)
(define-syntax video-module-begin
  (make-wrapping-module-begin ...))
```

```
(require syntax/wrapping-modbeg)
(define-syntax video-module-begin
  (make-wrapping-module-begin ...))
```

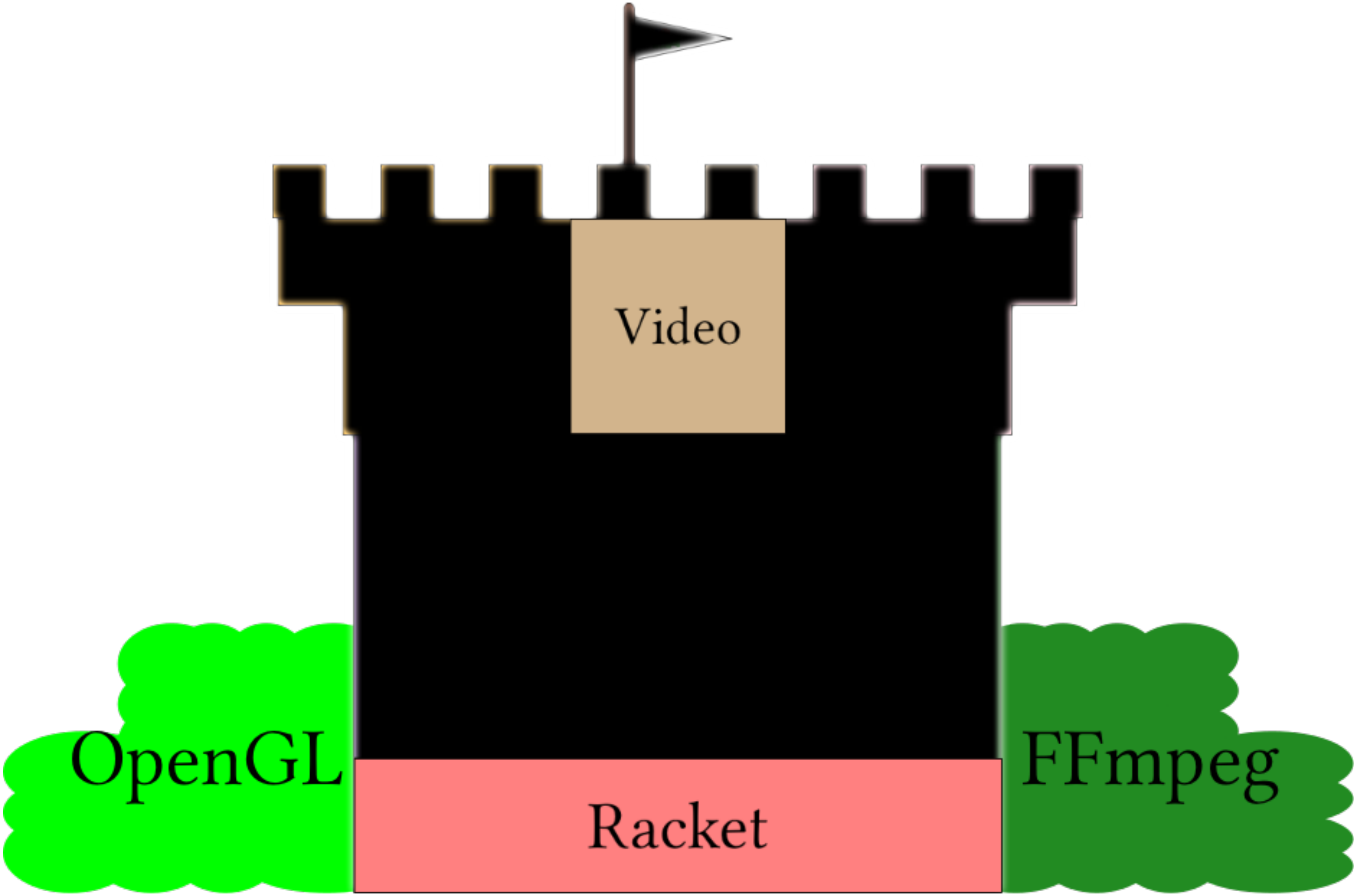
```
#lang racket/base
... run time code ...

(define-syntax macro-name
  ... compile time code ...)

... run time code ...))
```

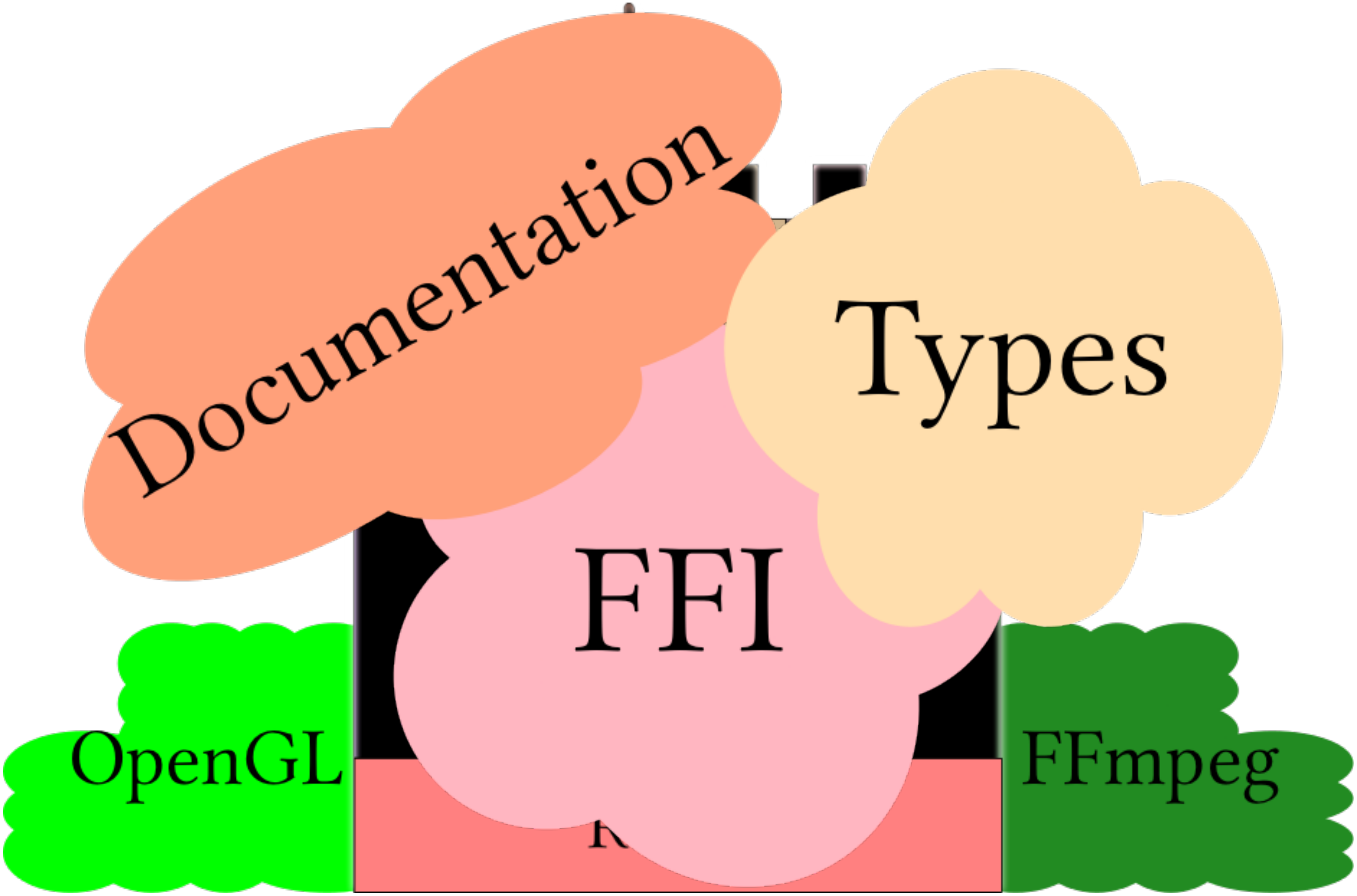
```
(define-syntax id expr)
```

`id` : run time binding

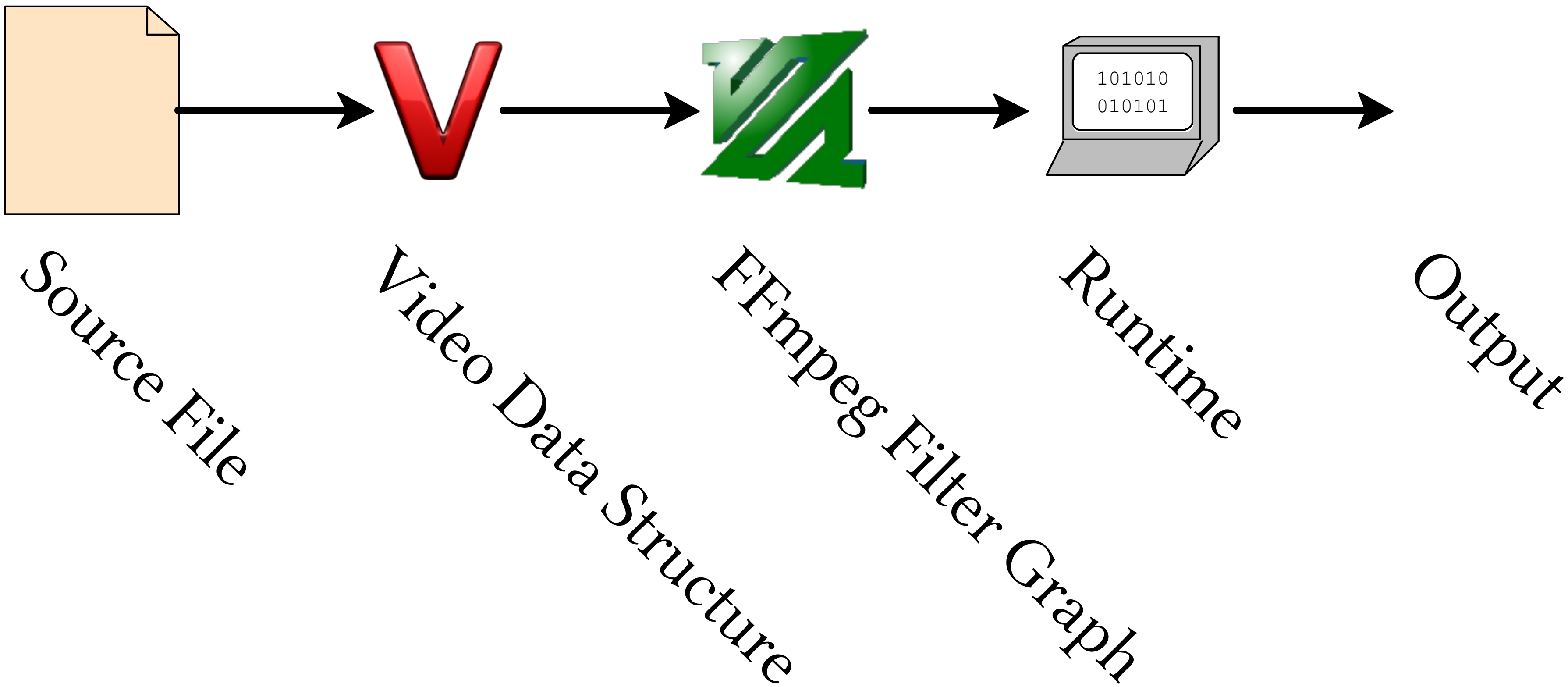
`expr` : compile time expression

Movies as Programs: A Tower of Languages


Video
OpenGL
FFmpeg
Racket


Documentation
Types
FFI
OpenGL
FFmpeg

FFI


101010
010101
Source File
Video Data Structure
FFmpeg Filter Graph
Runtime
Output

We have a FFI problem…

We have a FFI problem...

We want to solve it in the problem domain's own language...

We have a FFI problem…

We want to solve it in the problem domain's own language…

Make a DSL!

An FFI DSL

```
int av_frame_get_buffer(AVFrame *frame,
                        int align);
```

(Scheme Workshop, 2004)

An FFI DSL

```
int av_frame_get_buffer(AVFrame *frame,
                        int align);
```

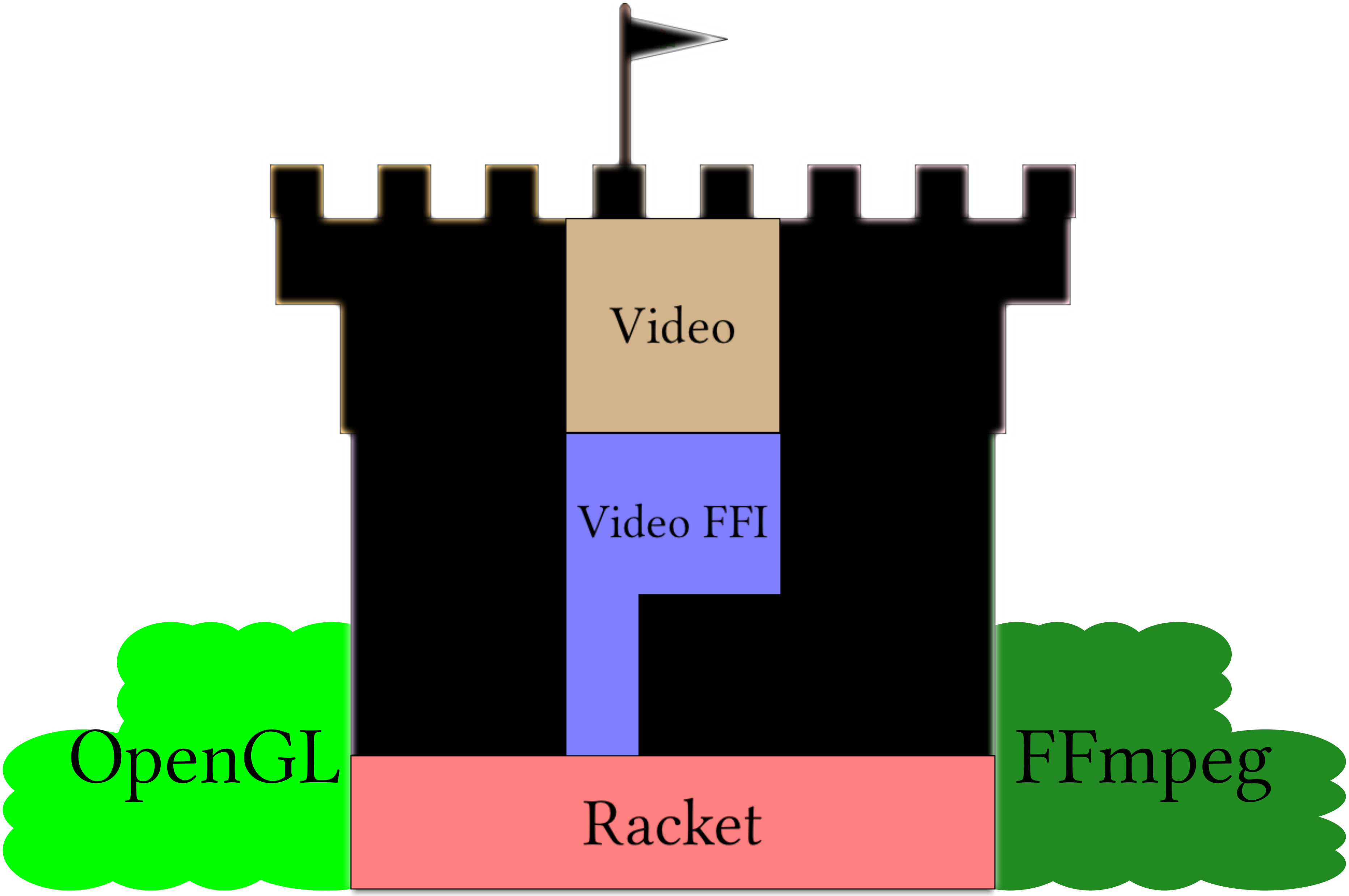
```
(define-ffmpeg av-frame-get-buffer
  (_fun [frame : _av-frame] [align : _int]
        -> [ret : _int]
        -> (maybe-error? ret)))
```

(Scheme Workshop, 2004)

An Object DSL

```
(define-ffmpeg av-frame-alloc ...)
(define-ffmpeg av-frame-free ...)
```

```
(define-constructor clip video
  ... av-frame-alloc ...
      av-frame-free ...)
```


Video
Video FFI
OpenGL
FFmpeg
Racket

Documentation

Documentation

We have a problem…

DOCUMENTATION

We have a problem...

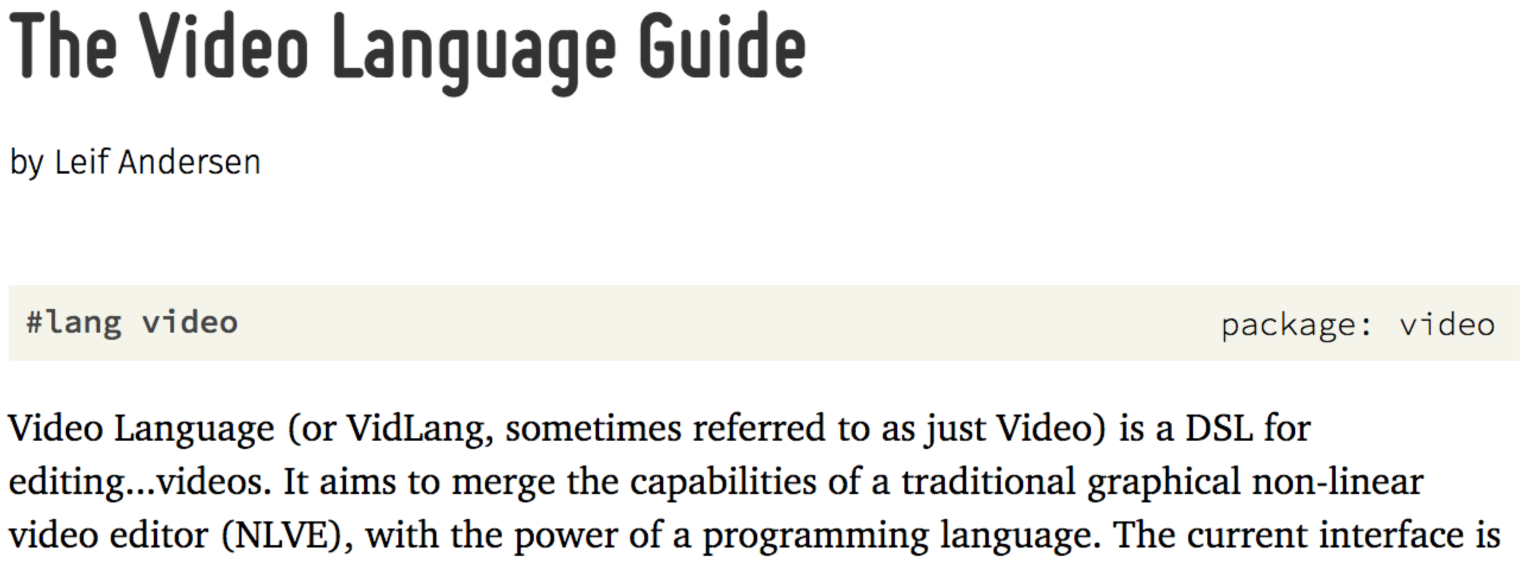
We want to solve it in the problem domain's own language...

We have a DOCUMENTATION problem...

We want to solve it in the problem domain's own language...

Make a DSL!

A Documentation DSL



The Video Language Guide

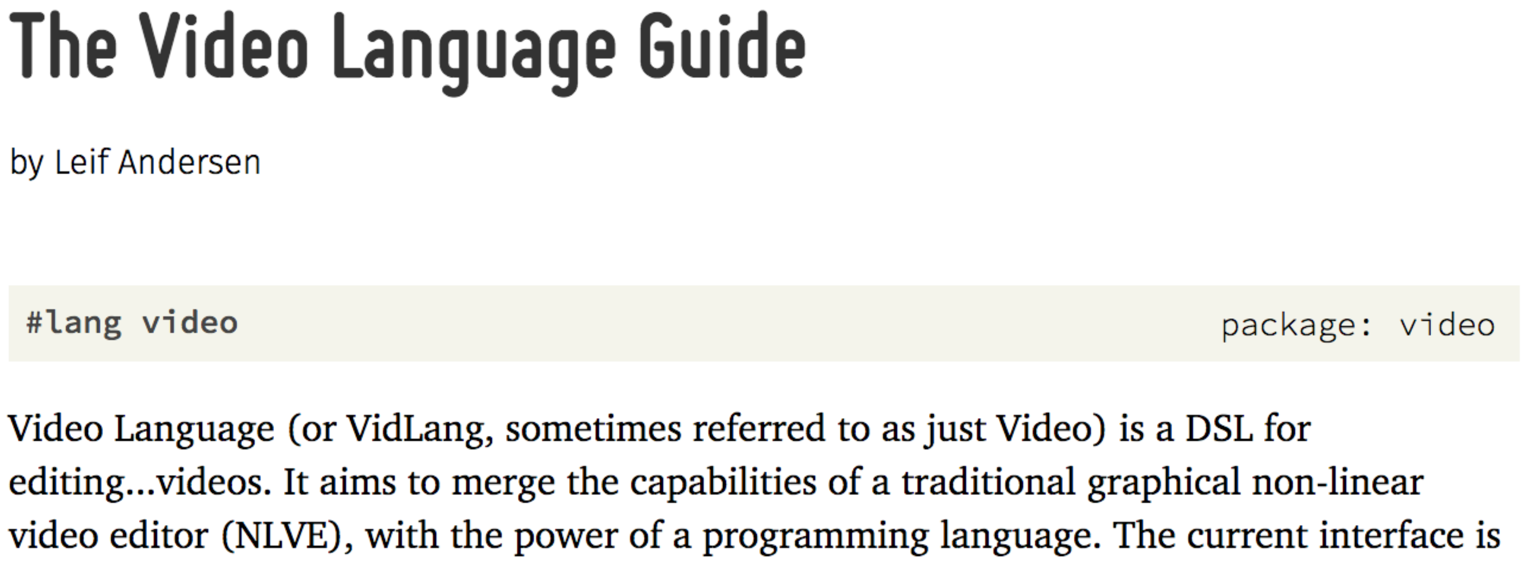
by Leif Andersen

```
#lang video                                                package: video
```

Video Language (or VidLang, sometimes referred to as just Video) is a DSL for editing...videos. It aims to merge the capabilities of a traditional graphical non-linear video editor (NLVE), with the power of a programming language. The current interface is

(ICFP, 2009)

A Documentation DSL



The Video Language Guide

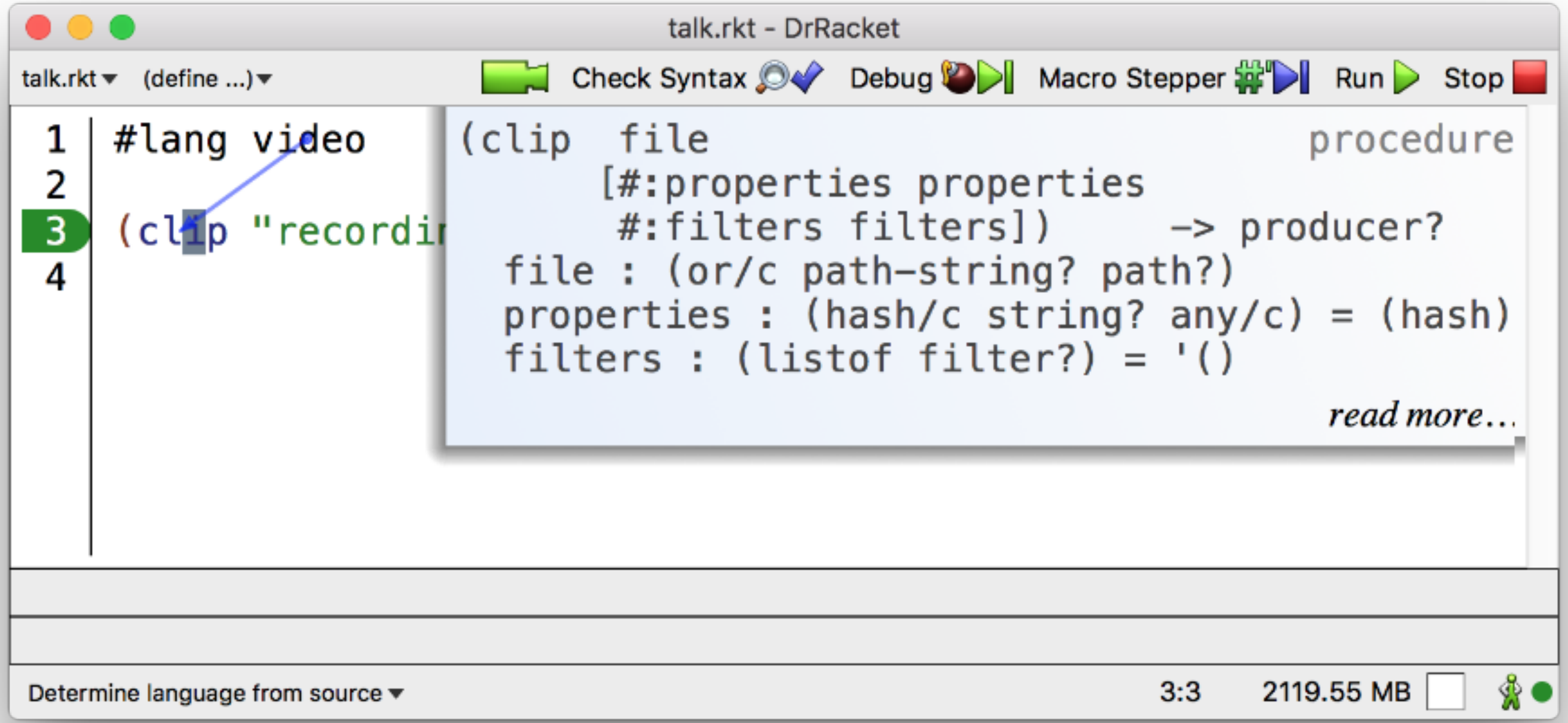
by Leif Andersen

#lang video package: video

Video Language (or VidLang, sometimes referred to as just Video) is a DSL for editing...videos. It aims to merge the capabilities of a traditional graphical non-linear video editor (NLVE), with the power of a programming language. The current interface is

```
#lang video/documentation
@title{Video: The Language}
@(defmodulelang video)

Video Language (or VidLang, sometimes referred
to as just Video) is a DSL for editing...videos.
It aims to merge the capabilities of a traditional
```

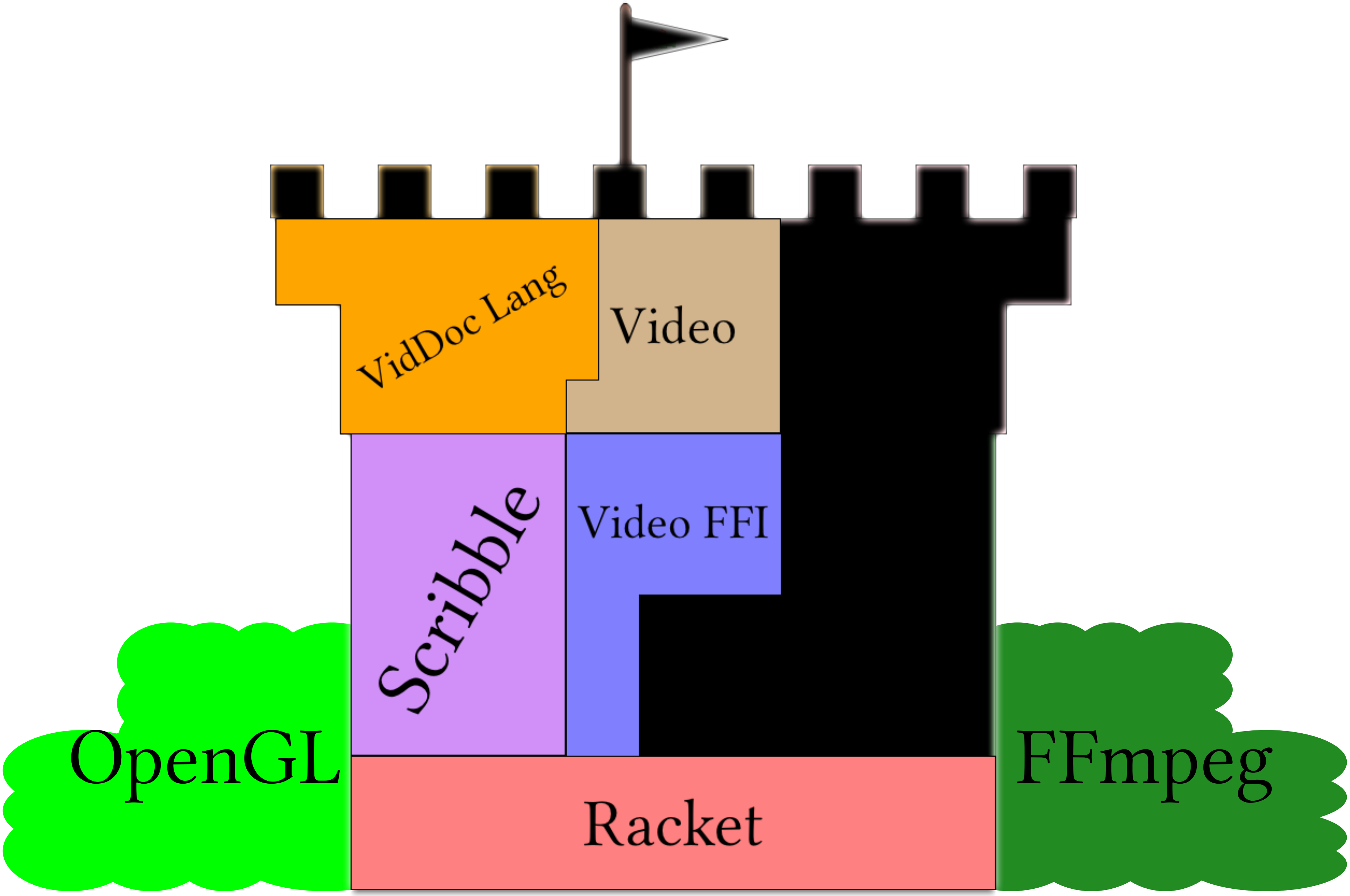
(ICFP, 2009)



```
#lang video

(clip "recordi
```

```
(clip  file                                        procedure
      [#:properties properties
       #:filters filters])      -> producer?
  file : (or/c path-string? path?)
  properties : (hash/c string? any/c) = (hash)
  filters : (listof filter?) = '()
```

read more...


VidDoc Lang
Video
Scribble
Video FFI
OpenGL
FFmpeg
Racket

Types

```
(clip "clip.mp4"
      #:start 0
      #:end 50)
```

```
(cut-producer (clip "clip.mp4"
                    #:start 0
                    #:end 50)
              #:start 0
              #:end 100)
```

```
(cut-produc   (cl    "clip.mp4"
                     #:start 0
                     #:end 50)
                    t 0
              #:e   d 100)
```

A Typed DSL

$$\frac{\texttt{m >= n}}{\texttt{(Producer m) <: (Producer n)}}$$

TYPES

We have a ∨ problem...

TYPES

We have a problem...

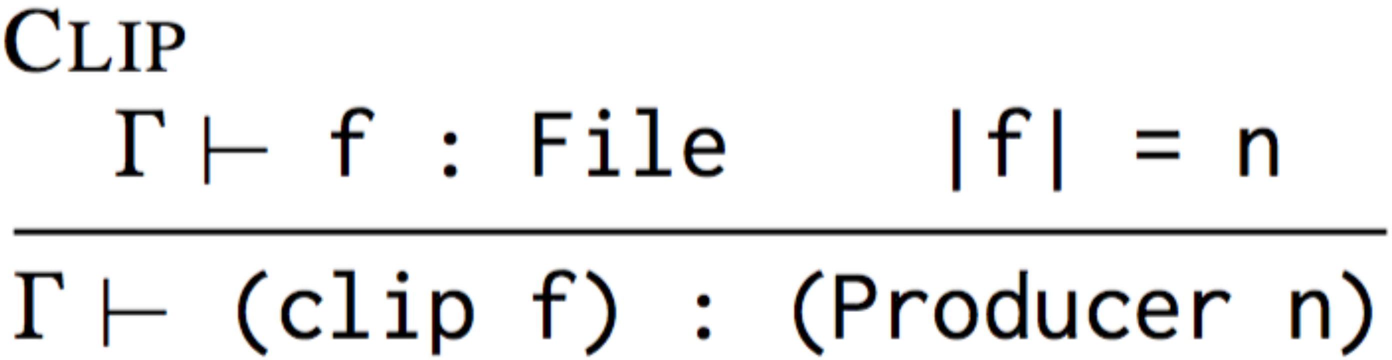
We want to solve it in the problem domain's own language...

We have a TYPES problem...

We want to solve it in the problem domain's own language...

Make a DSL!

A Typed DSL



Clip

$$\frac{\Gamma \vdash \texttt{f : File} \qquad |\texttt{f}| = \texttt{n}}{\Gamma \vdash \texttt{(clip f) : (Producer n)}}$$

(POPL, 2017)

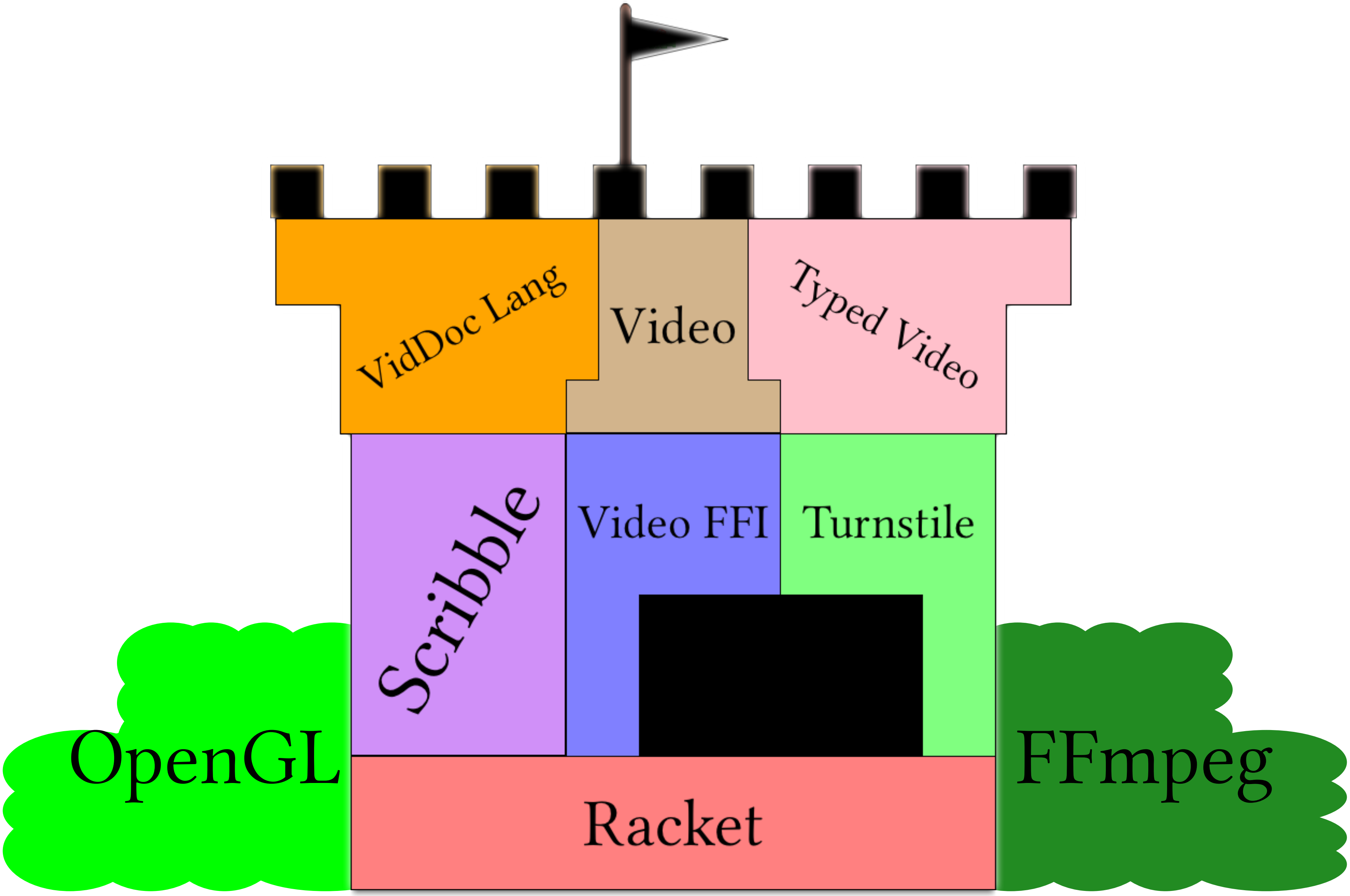
A Type Implementation DSL

Clip

$$\frac{\Gamma \vdash \texttt{f : File} \qquad \texttt{|f| = n}}{\Gamma \vdash \texttt{(clip f) : (Producer n)}}$$

```
(define-typed-syntax (clip f) ≫
  [⊢ f ≫ _ ⇐ File] #:where n (length f)
  -------------------------------------
  [⊢ (untyped:clip f) ⇒ (Producer n)])
```

(POPL, 2017)


VidDoc Lang
Video
Typed Video
Scribble
Video FFI
Turnstile
OpenGL
Racket
FFmpeg

We have a DSL problem...

We have a DSL problem...

We want to solve it in the problem domain's own language...

We have a DSL problem...

We want to solve it in the problem domain's own language...

syntax-parse

A DSL for making DSLs

(ICFP, 2010)

```
(define-syntax-rule
  (define/playlist (name args ...)
    body ...)
  (define name
    (λ (args ...)
      (playlist body ...))))
```

```
(define-syntax-rule
  (define/playlist (name args ...)
   body ...)
  (define name
    (λ (args ...)
      (playlist body ...))))
```

```
> (define/playlist (double A)
    A
    A)
```

```
(define-syntax-rule
  (define/playlist (name args ...)
    body ...)
  (define name
    (λ (args ...)
      (playlist body ...))))
```

```
> (define/playlist (double (A B C))
    A)
```

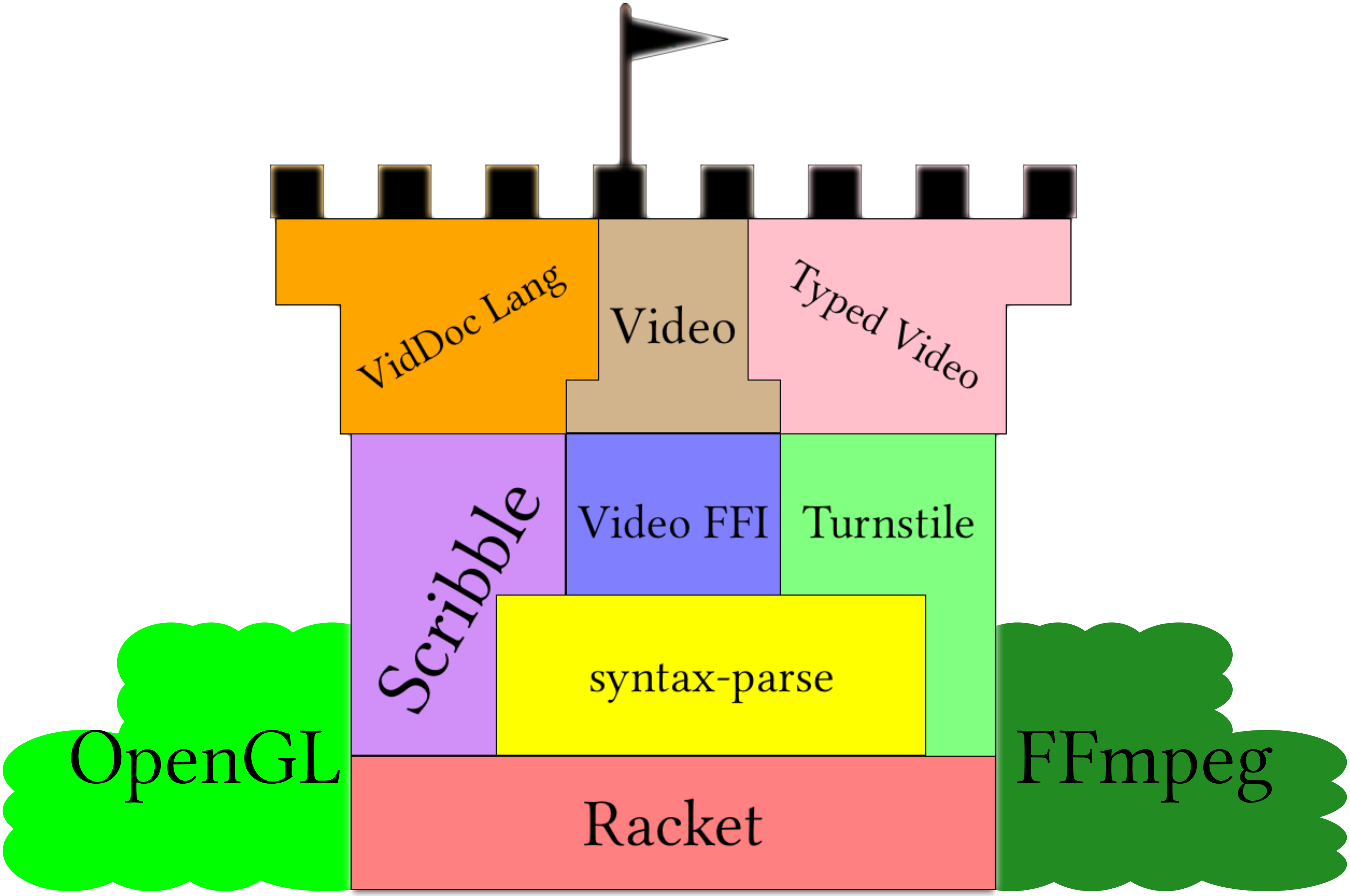
```
(define-simple-macro
  (define/playlist header:function-header
    body ...)
  (define header.name
    (λ header.args
      (playlist body ...))))
```

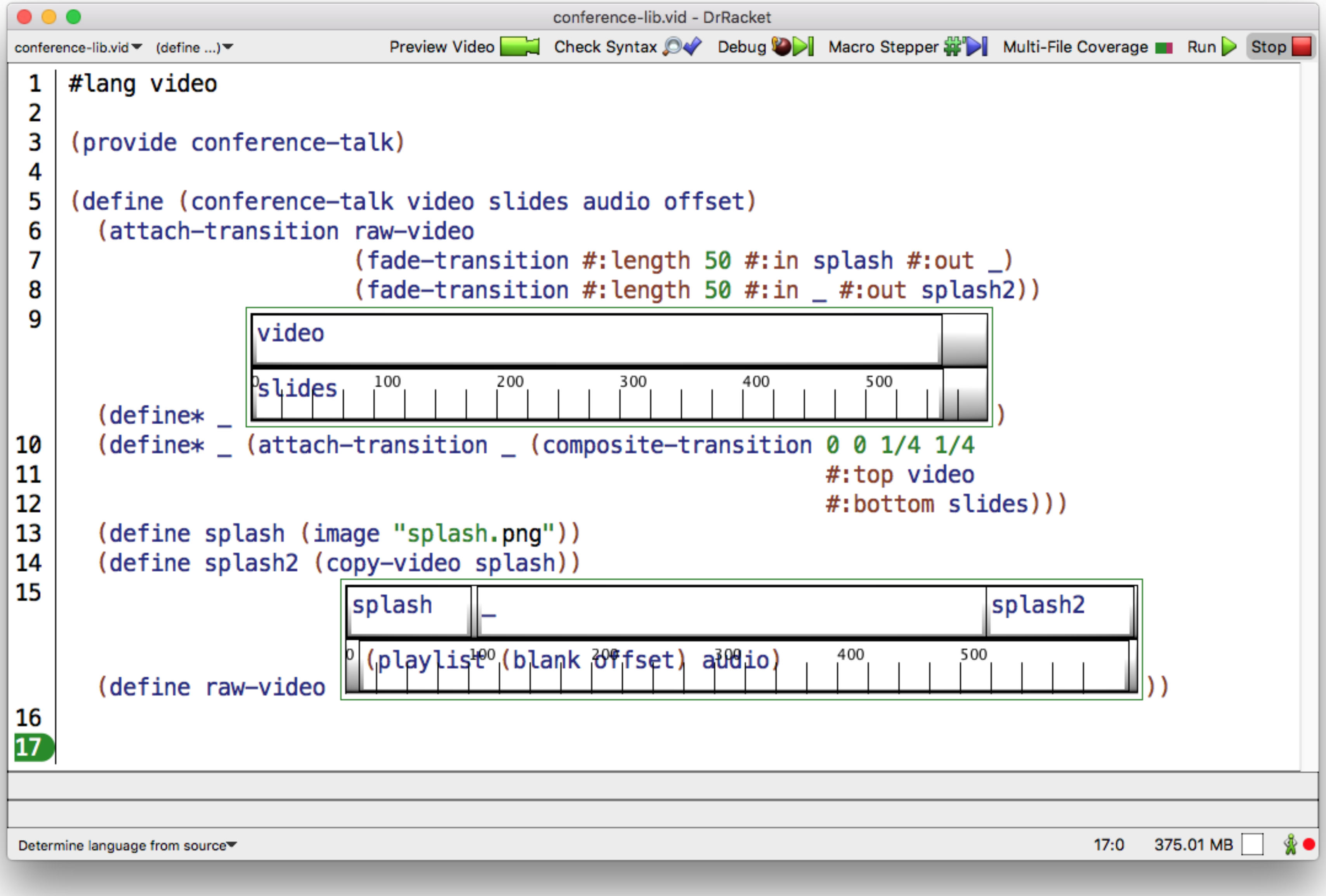
```
(define-simple-macro
  (define/playlist header:function-header
    body ...)
  (define header.name
    (λ header.args
      (playlist body ...))))
```

```
> (define/playlist (double A)
   A
   A)
```

```
(define-simple-macro
  (define/playlist header:function-header
    body ...)
  (define header.name
    (λ header.args
      (playlist body ...))))
```

```
> (define/playlist (double (A B C))
    A)
```


VidDoc Lang
Video
Typed Video
Scribble
Video FFI
Turnstile
syntax-parse
OpenGL
Racket
FFmpeg



```
#lang video

(provide conference-talk)

(define (conference-talk video slides audio offset)
  (attach-transition raw-video
                     (fade-transition #:length 50 #:in splash #:out _)
                     (fade-transition #:length 50 #:in _ #:out splash2))
                 video
                 slides
  (define* _                                                             )
  (define* _ (attach-transition _ (composite-transition 0 0 1/4 1/4
                                                        #:top video
                                                        #:bottom slides)))
  (define splash (image "splash.png"))
  (define splash2 (copy-video splash))
                        splash   _                              splash2
  (define raw-video      (playlist (blank offset) audio)                  ))

```

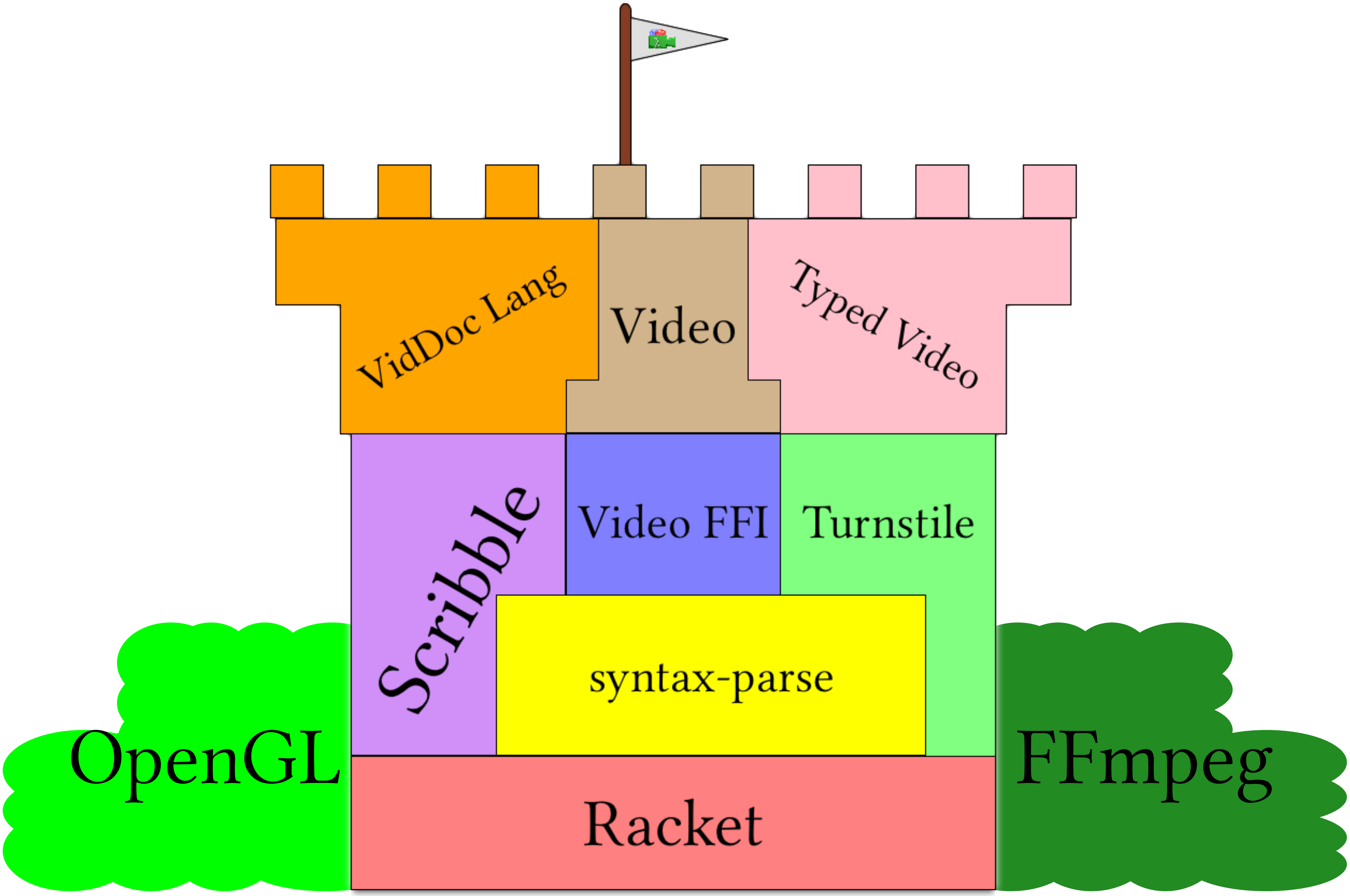
Editor-Oriented Programming

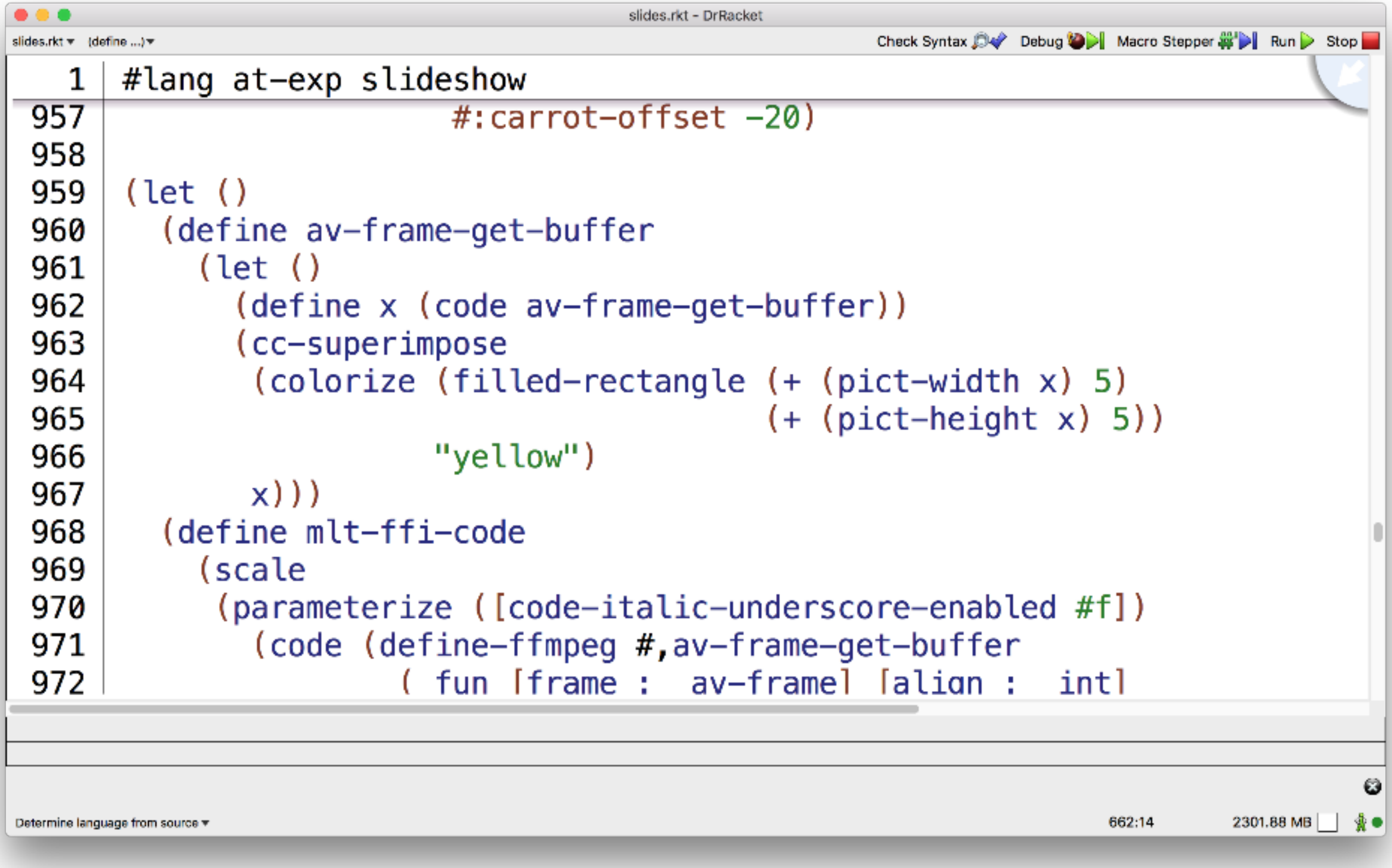
The Future…

```
begin-for-syntax

define-syntax
```

```
begin-for-editor

define-editor
```

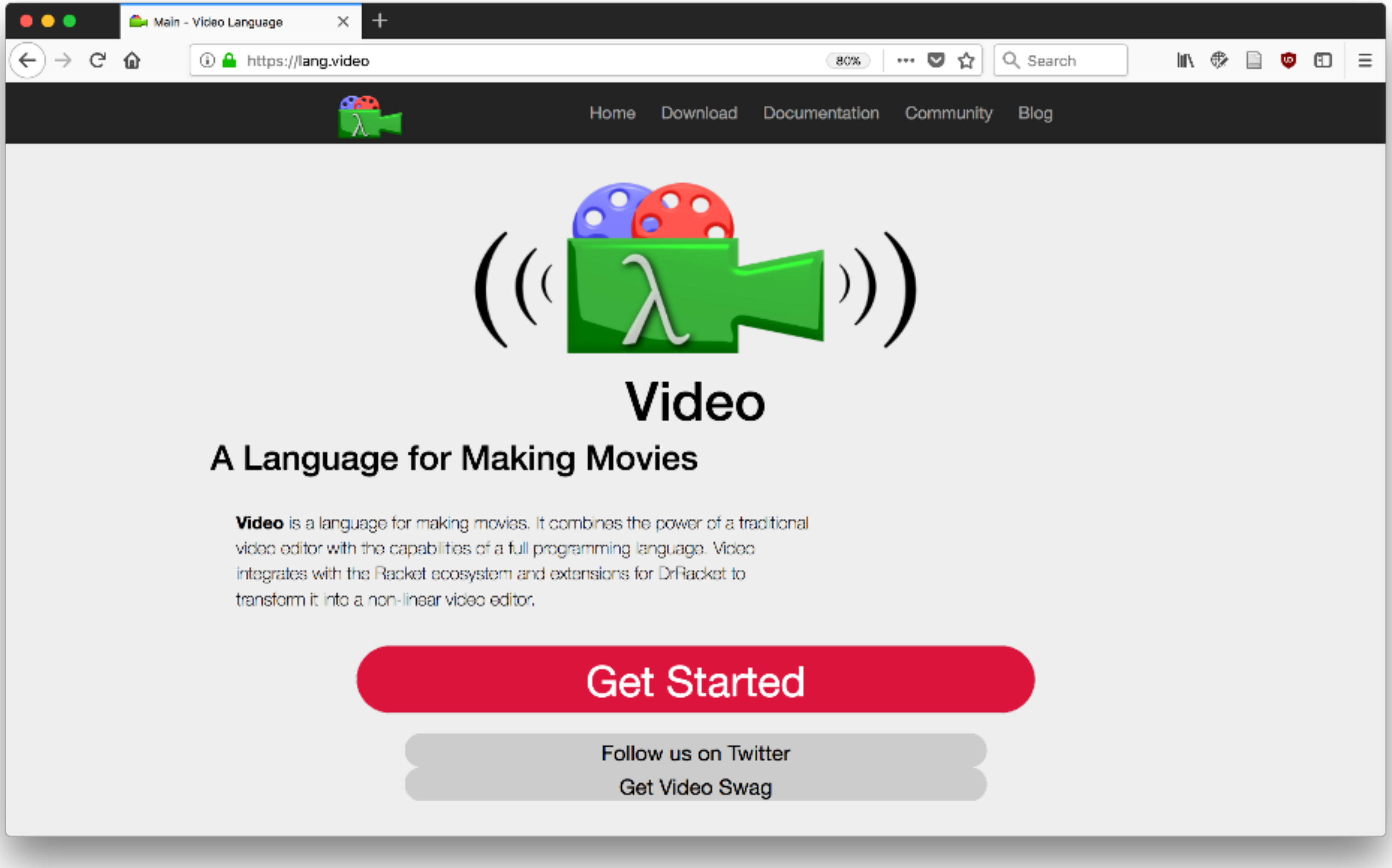
```
#lang editor

(define-editor video-editor ...)

...






(play                                   )
```


VidDoc Lang
Video
Typed Video
Scribble
Video FFI
Turnstile
syntax-parse
OpenGL
FFmpeg
Racket



```
#lang at-exp slideshow
                  #:carrot-offset -20)

(let ()
  (define av-frame-get-buffer
    (let ()
      (define x (code av-frame-get-buffer))
      (cc-superimpose
       (colorize (filled-rectangle (+ (pict-width x) 5)
                                   (+ (pict-height x) 5))
                 "yellow")
       x)))
  (define mlt-ffi-code
    (scale
     (parameterize ([code-italic-underscore-enabled #f])
       (code (define-ffmpeg #,av-frame-get-buffer
               ( fun [frame :  av-frame] [align :  int]
```

https://lang.video

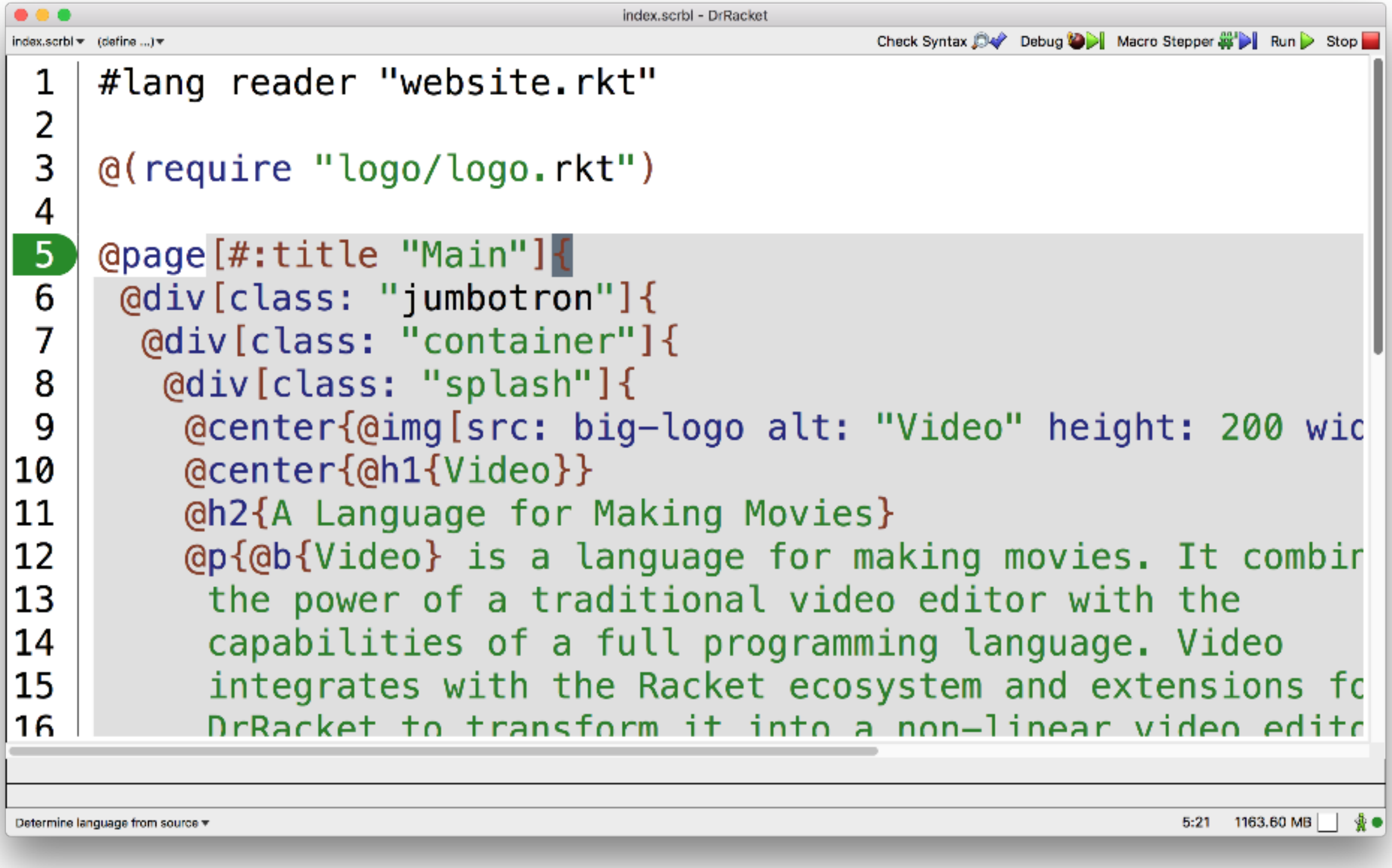


Video

A Language for Making Movies

Video is a language for making movies. It combines the power of a traditional video editor with the capabilities of a full programming language. Video integrates with the Racket ecosystem and extensions for DrRacket to transform it into a non-linear video editor.

https://lang.video



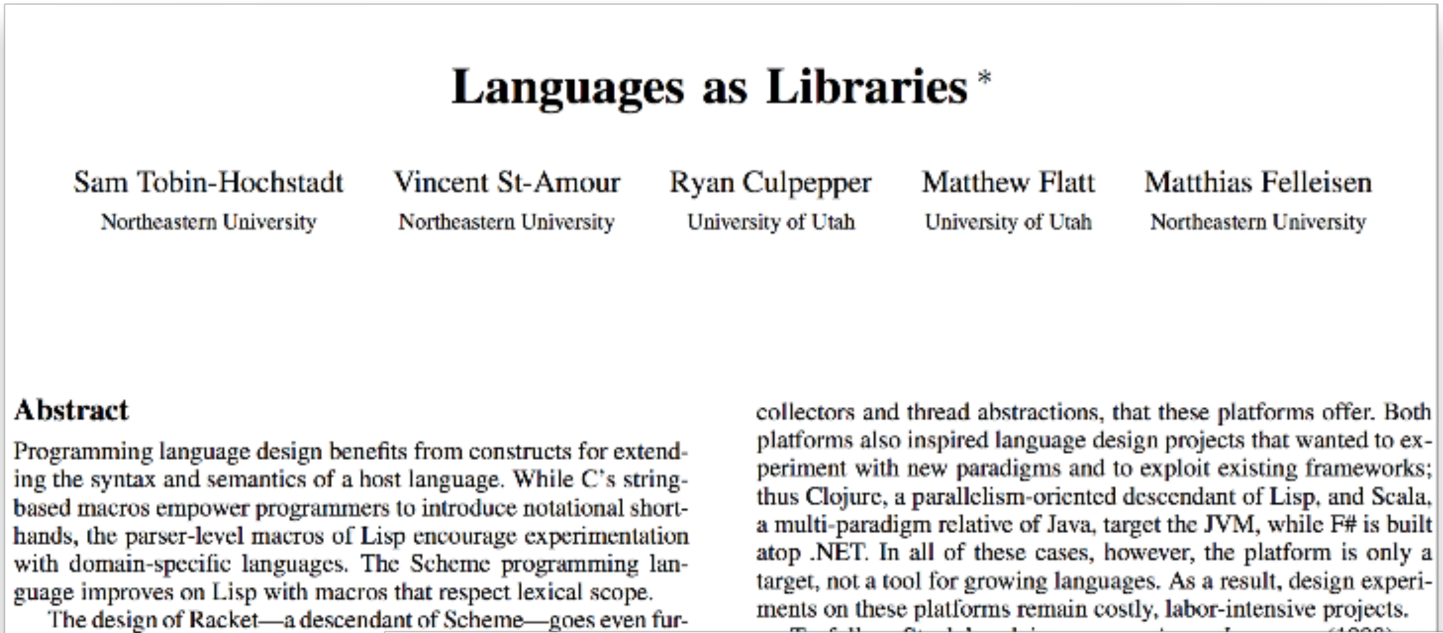
```
#lang reader "website.rkt"

@(require "logo/logo.rkt")

@page[#:title "Main"]{
 @div[class: "jumbotron"]{
  @div[class: "container"]{
   @div[class: "splash"]{
    @center{@img[src: big-logo alt: "Video" height: 200 wic
    @center{@h1{Video}}
    @h2{A Language for Making Movies}
    @p{@b{Video} is a language for making movies. It combir
     the power of a traditional video editor with the
     capabilities of a full programming language. Video
     integrates with the Racket ecosystem and extensions fc
     DrRacket to transform it into a non-linear video editc
```

BEAUTIFUL RACKET

HOW TO MAKE YOUR OWN
PROGRAMMING LANGUAGES WITH RACKET
BY MATTHEW BUTTERICK · VERSION 1.1

Fear of Macros

by Greg Hendershott



Languages as Libraries *

Sam Tobin-Hochstadt
Northeastern University

Vincent St-Amour
Northeastern University

Ryan Culpepper
University of Utah

Matthew Flatt
University of Utah

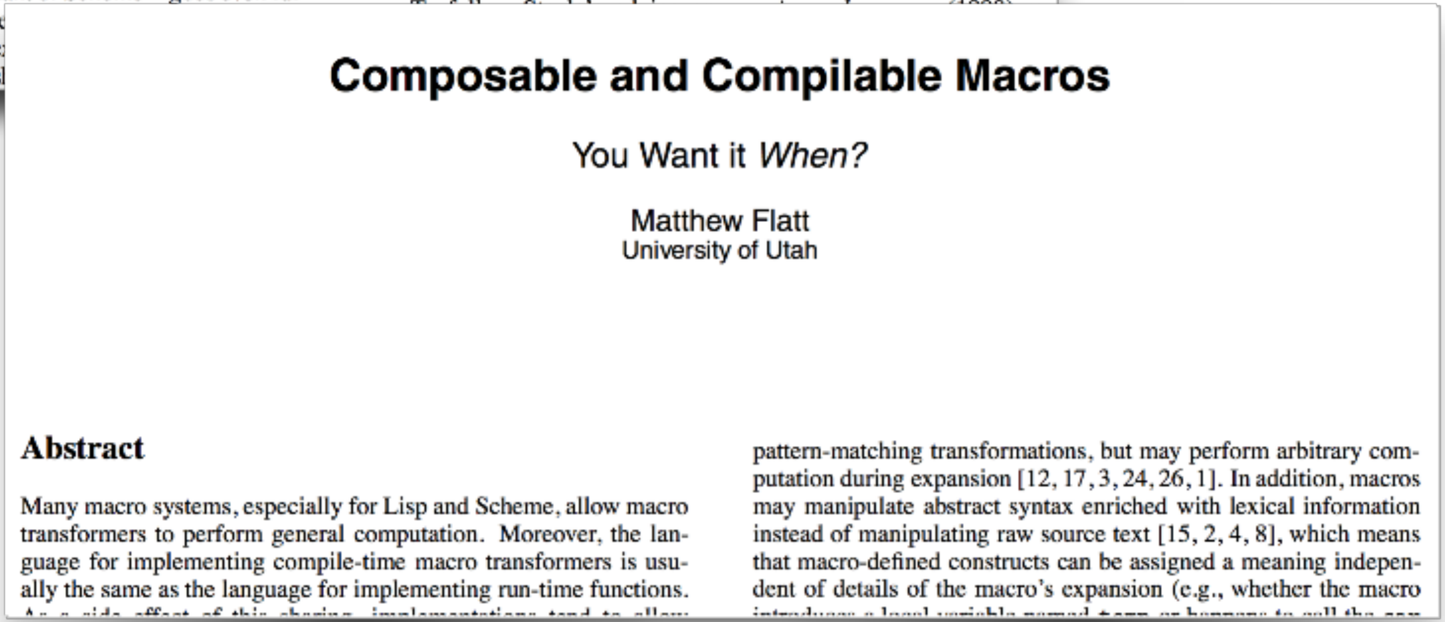
Matthias Felleisen
Northeastern University

Abstract

Programming language design benefits from constructs for extending the syntax and semantics of a host language. While C's string-based macros empower programmers to introduce notational shorthands, the parser-level macros of Lisp encourage experimentation with domain-specific languages. The Scheme programming language improves on Lisp with macros that respect lexical scope.

The design of Racket—a descendant of Scheme—goes even further with the introduction of a full-fle
mantics of the language. A Racket e
add constructs that are indistinguis

collectors and thread abstractions, that these platforms offer. Both platforms also inspired language design projects that wanted to experiment with new paradigms and to exploit existing frameworks; thus Clojure, a parallelism-oriented descendant of Lisp, and Scala, a multi-paradigm relative of Java, target the JVM, while F# is built atop .NET. In all of these cases, however, the platform is only a target, not a tool for growing languages. As a result, design experiments on these platforms remain costly, labor-intensive projects.



Composable and Compilable Macros

You Want it *When?*

Matthew Flatt
University of Utah

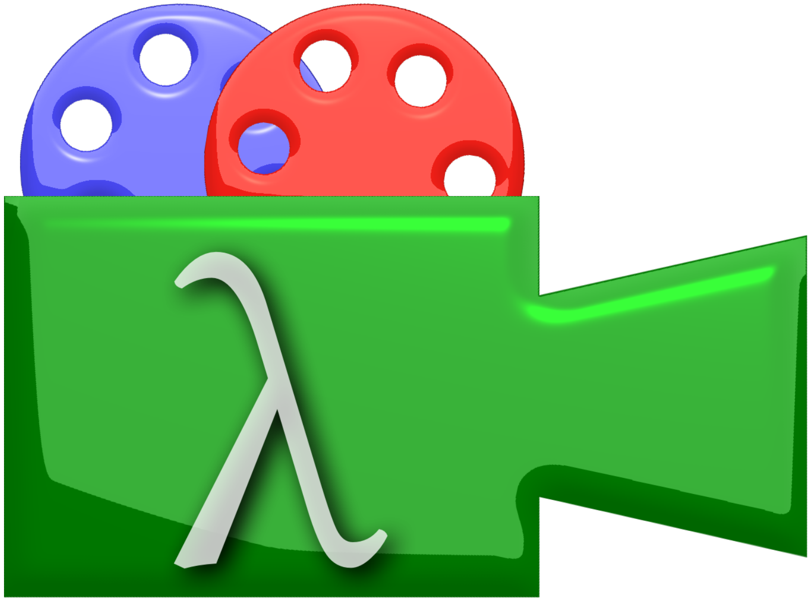
Abstract

Many macro systems, especially for Lisp and Scheme, allow macro transformers to perform general computation. Moreover, the language for implementing compile-time macro transformers is usually the same as the language for implementing run-time functions.

pattern-matching transformations, but may perform arbitrary computation during expansion [12, 17, 3, 24, 26, 1]. In addition, macros may manipulate abstract syntax enriched with lexical information instead of manipulating raw source text [15, 2, 4, 8], which means that macro-defined constructs can be assigned a meaning independent of details of the macro's expansion (e.g., whether the macro

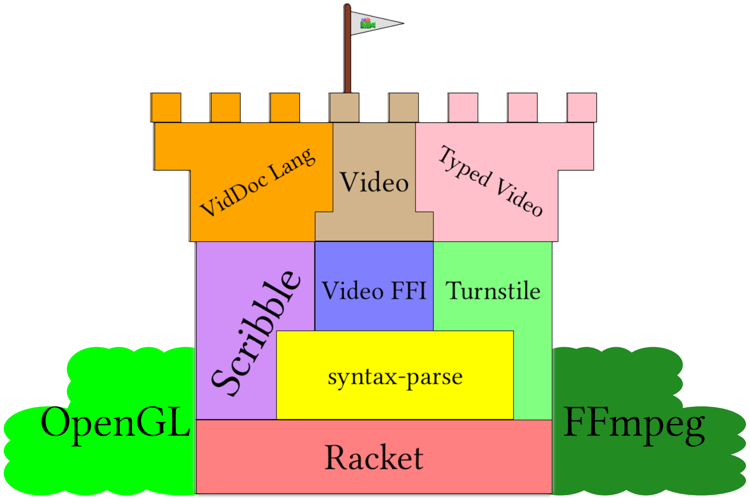
Thanks For Watching

`http://lang.video`
`@videolang`



We make DSLs using

Linguistic Inheritance


VidDoc Lang
Video
Typed Video
Scribble
Video FFI
Turnstile
syntax-parse
OpenGL
Racket
FFmpeg