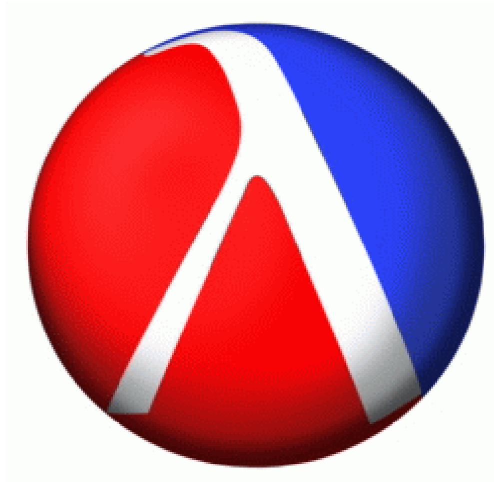
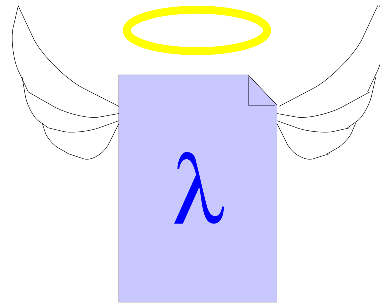
Programming Languages as Operating Systems



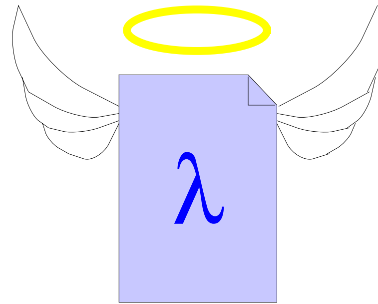
Matthew Flatt

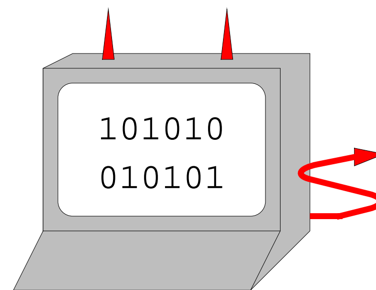
University of Utah

Programming in Heaven

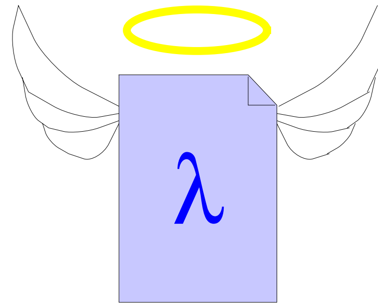


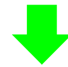
Programming in Heaven

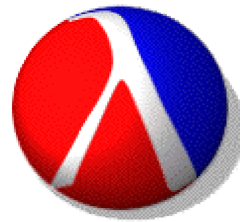




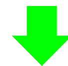
Programming in Heaven

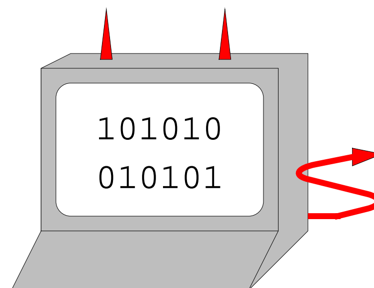




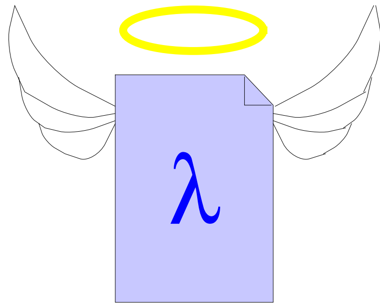


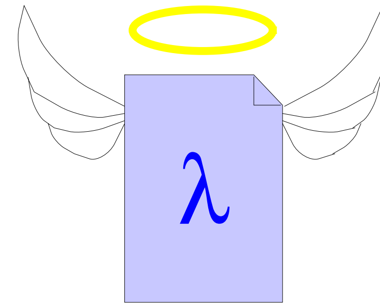
language run-time

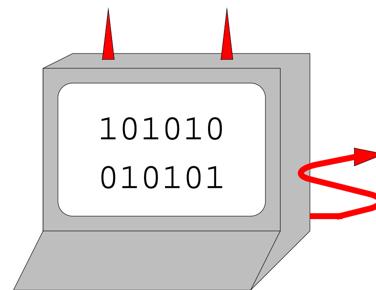




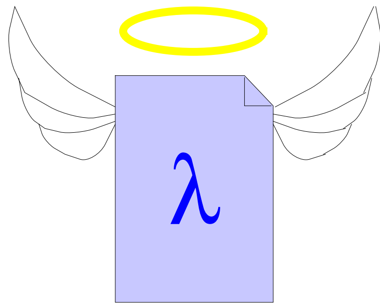
Multi-Programming

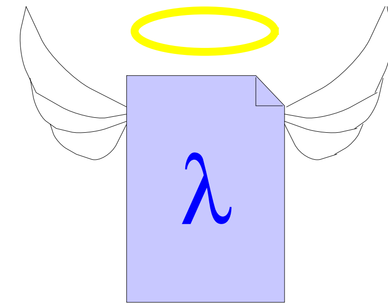


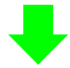


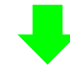


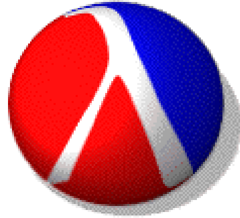
Multi-Programming

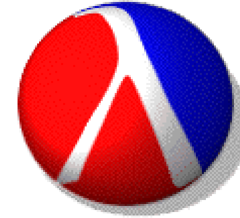


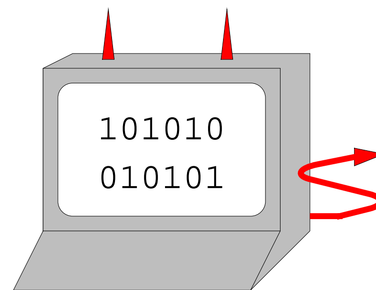




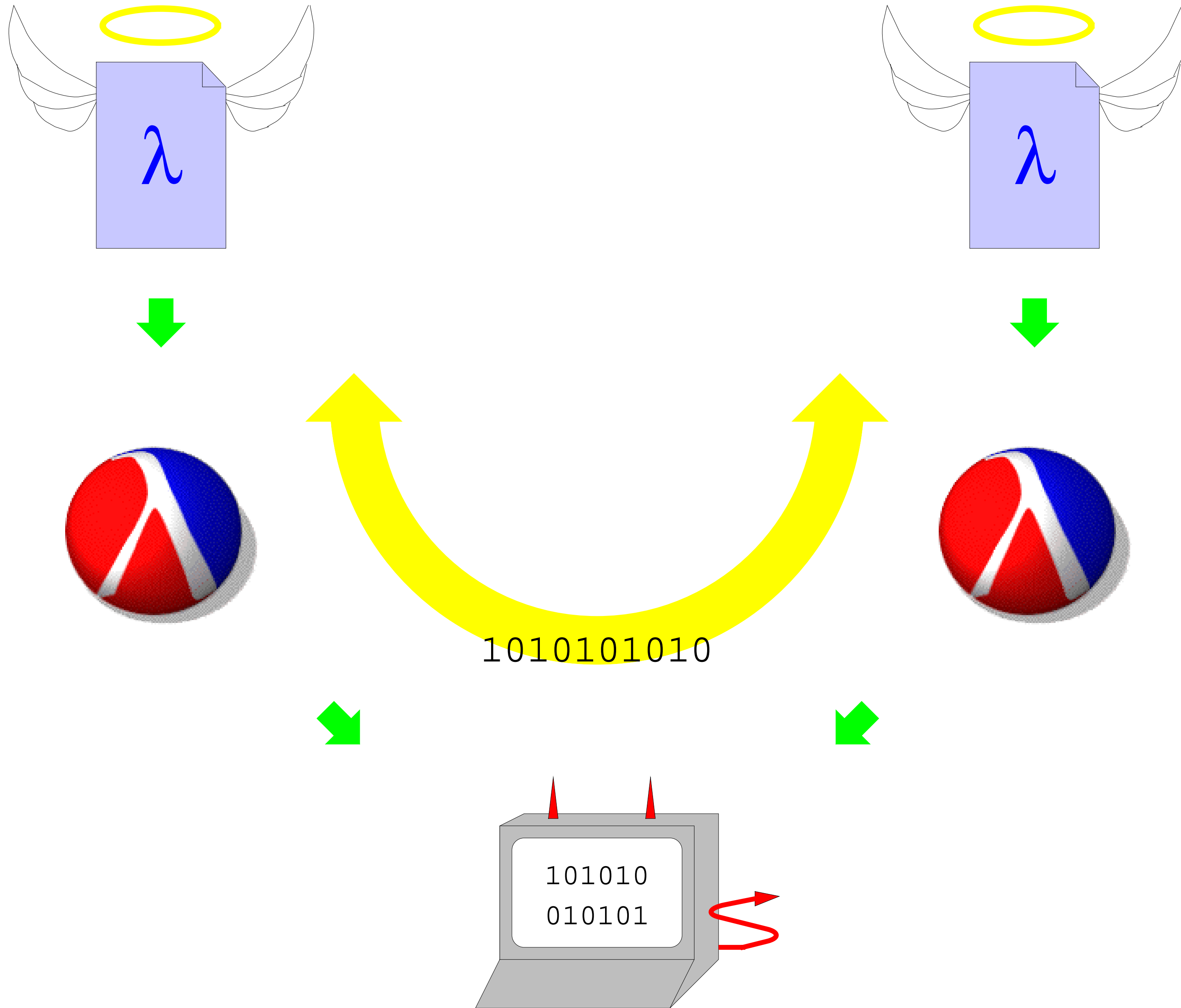




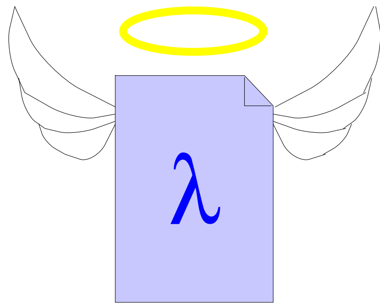
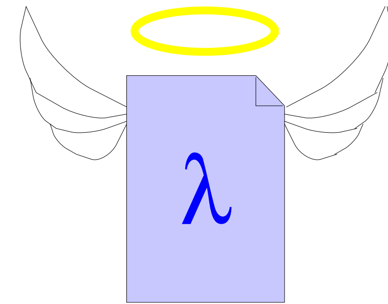
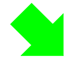


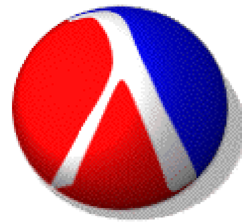
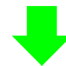
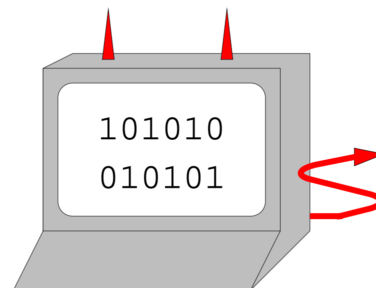


Multi-Programming

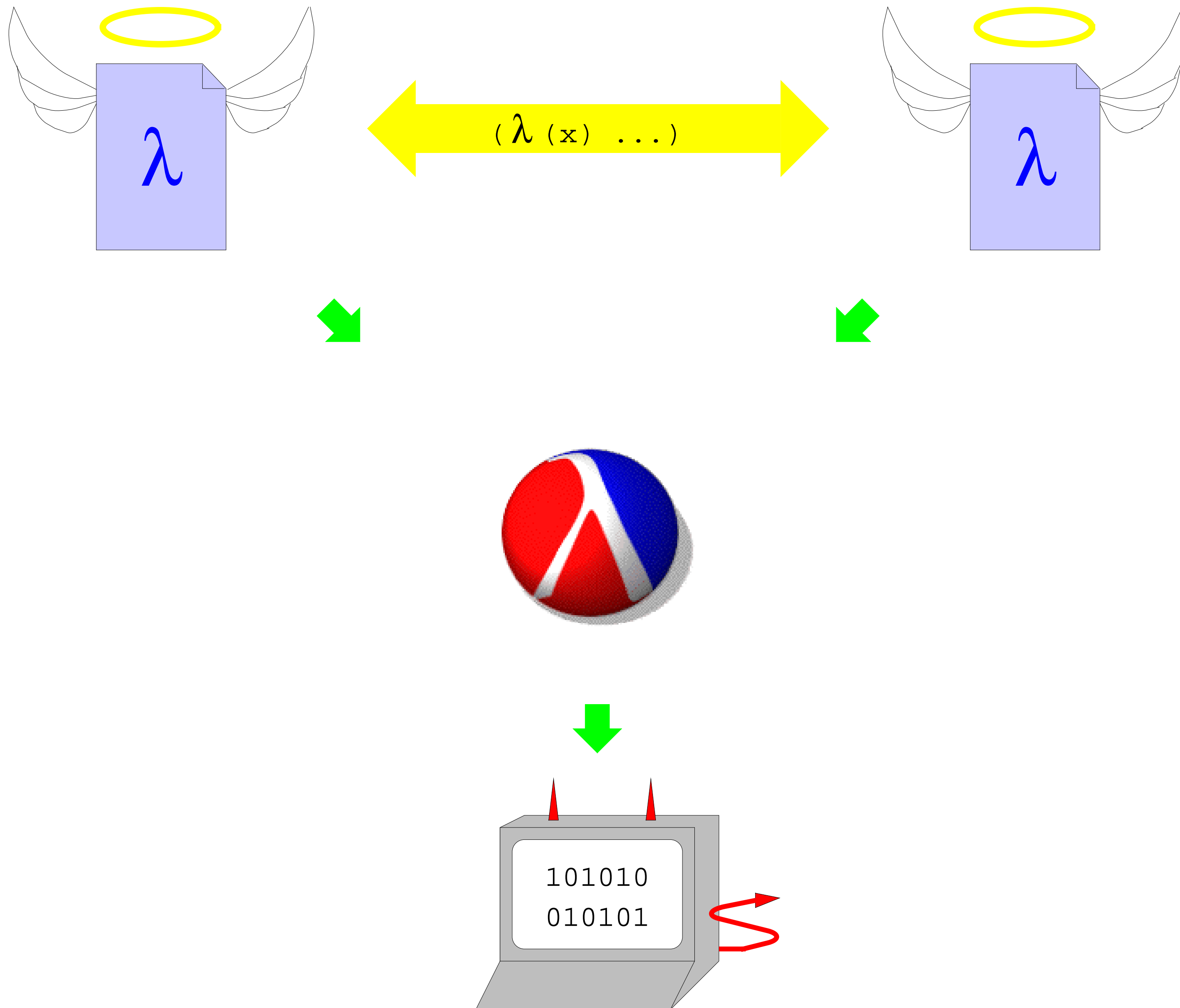


Multi-Programming in Heaven

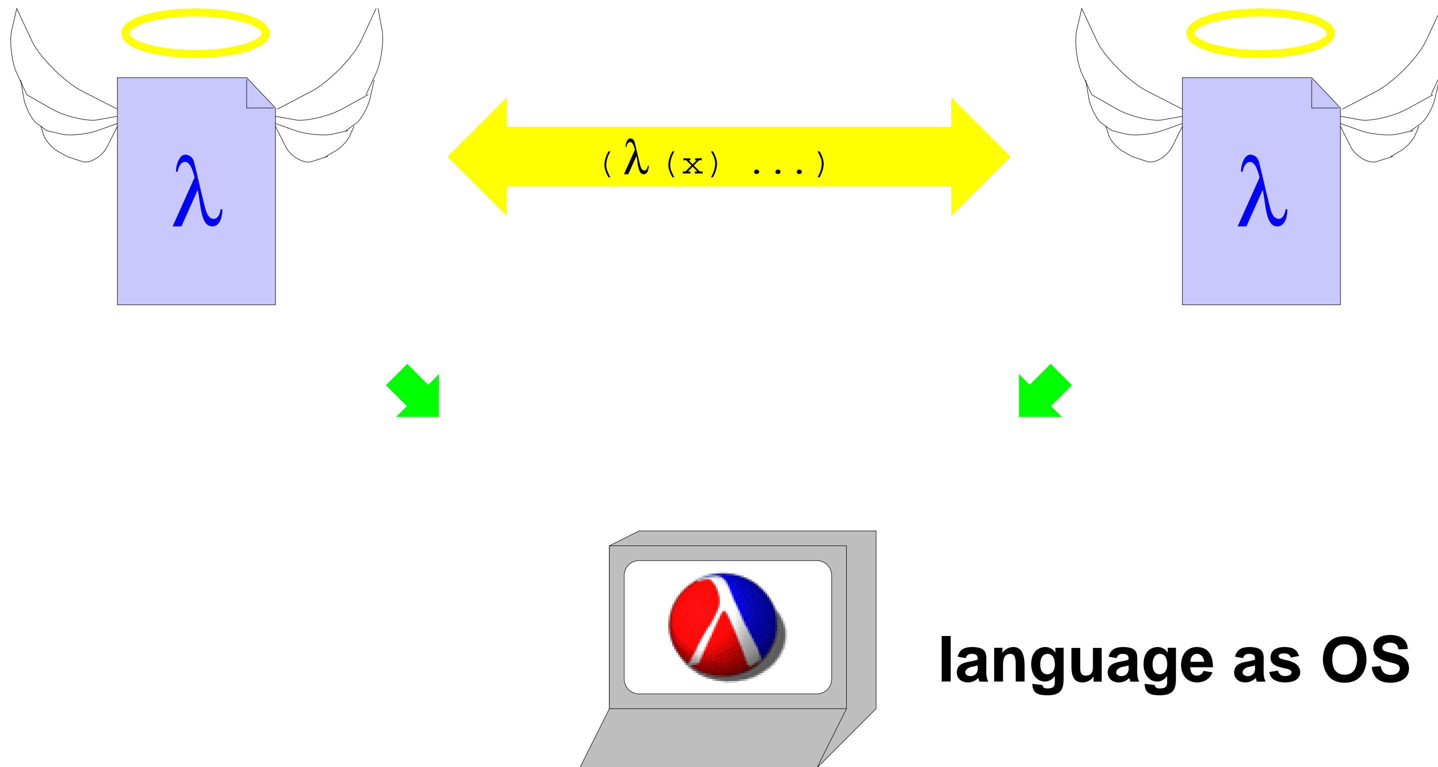




Multi-Programming in Heaven



Multi-Programming in Heaven



language as OS

Languages as Operating Systems

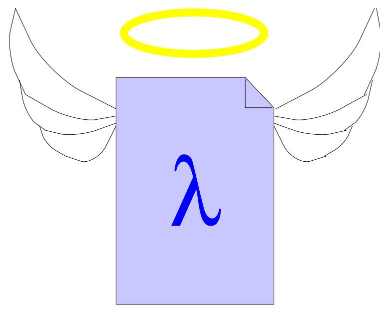
Language as OS $\Rightarrow$ process controls:

- Separate threads of evaluation
- Separate process-specific state (e.g., current directory)
- Separate graphical event loops
- Ability to terminate a process and reclaim its resources

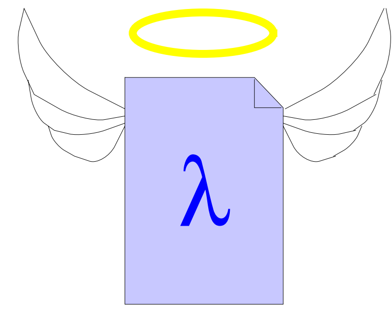
Language-based OSes: Pilot [Redell80], SPIN [Bershad95], ...

Extended languages: JKernel [Hawblitzel98], Alta [Tullman99], KaffeOS [Back00], ...

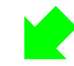
Example: Processes in a Language

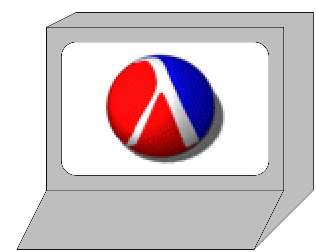


DrScheme



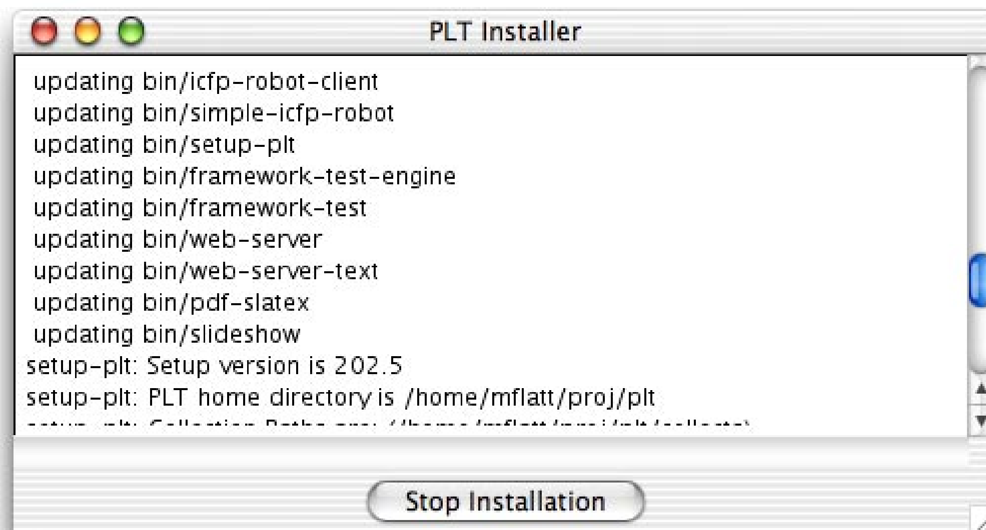
user's program



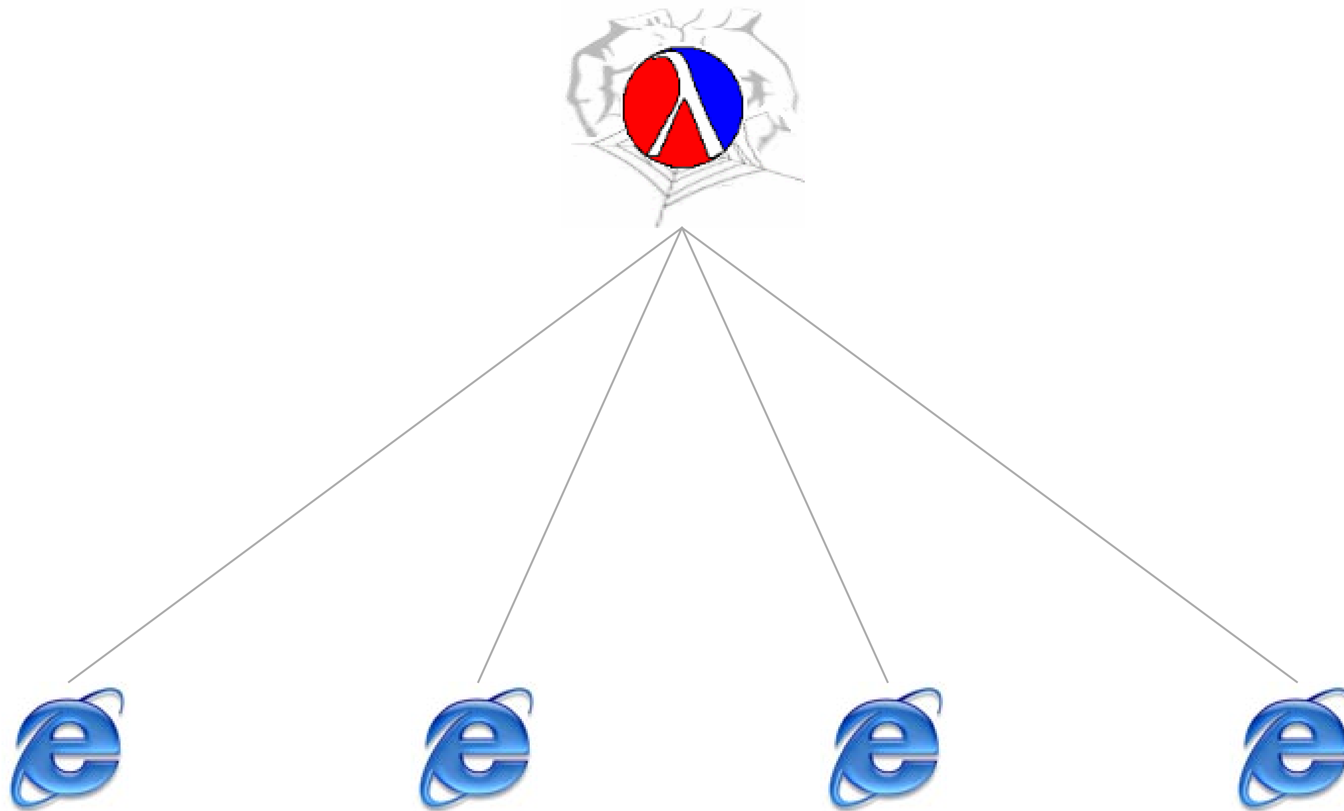


Run DrScheme

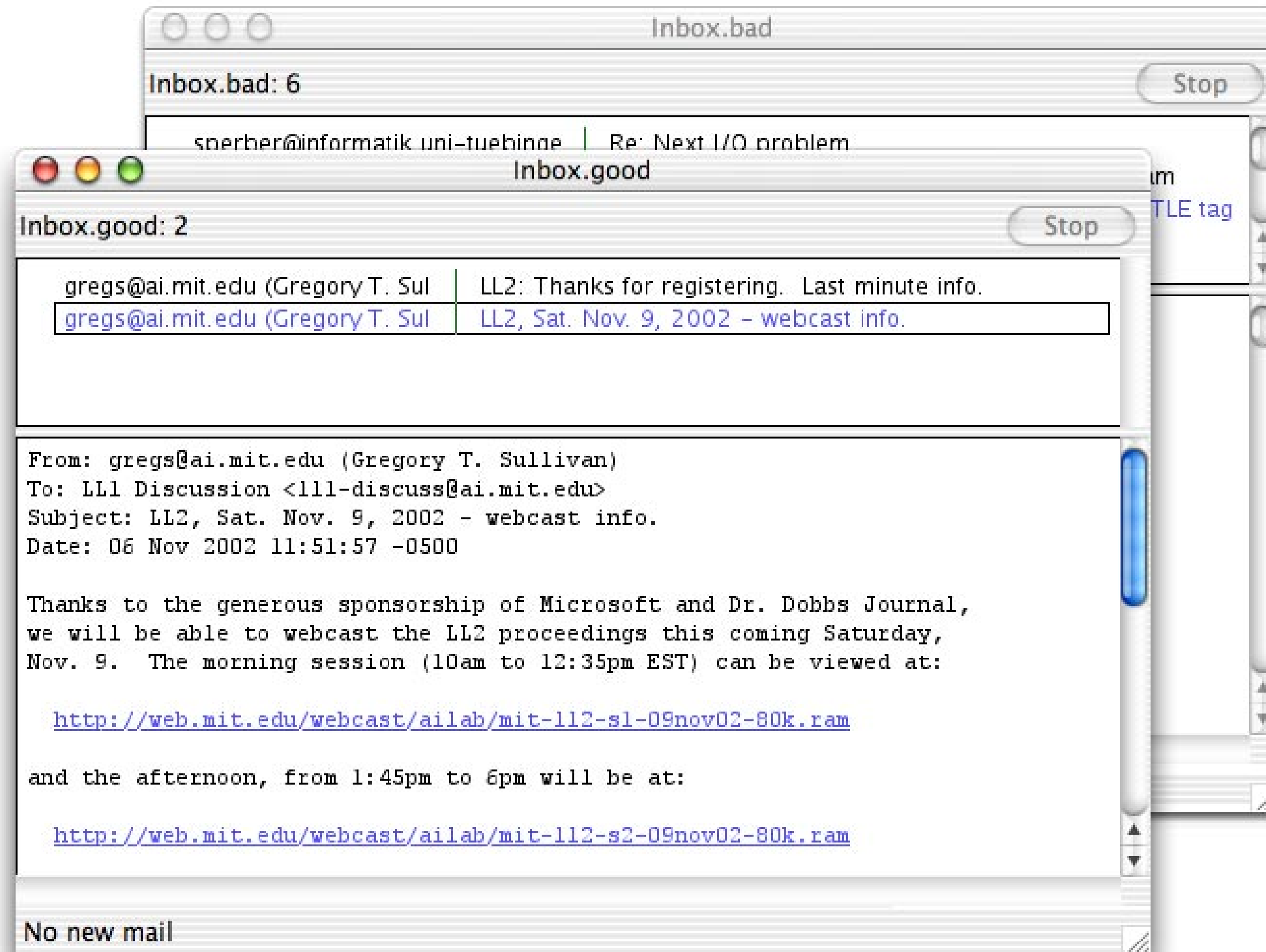
More Process Examples



More Process Examples



More Process Examples



- **Motivation and Approach**
- **PLT Scheme as an Operating System**
    - Threads
    - Parameters
    - Eventspaces
    - Custodians
- **Putting the Pieces to Work**

Threads

Concurrent execution

```
(require "spin-display.scm")
```
eval

```
(define (spin)
  (rotate-a-little)
  (sleep 0.1)
  (spin))
```

```
(define spinner (thread spin))
```
eval

```
(kill-thread spinner)
```
eval

Parameters (a.k.a. Fluid Variables)

Thread-local state

```
(printf "Hello\n")
(fprintf (current-output-port) "Hola\n")
(fprintf (current-error-port) "Goodbye\n")
(error "Ciao")
```

eval

```
(parameterize ((current-error-port (current-output-port))
  (error "Au Revoir"))
```

eval

```
(parameterize ((current-error-port (current-output-port))
  (thread
    (lambda ()
      (error "Zai Jian"))))
```

eval

Eventspaces

Concurrent GUIs

```
(thread (lambda () (message-box "One" "Hi")))
(thread (lambda () (message-box "Two" "Bye")))
```

eval

```
(thread (lambda () (message-box "One" "Hi")))
(parameterize ((current-eventspace (make-eventspace)))
  (thread (lambda () (message-box "Two" "Bye"))))
```

eval

Custodians

Termination and clean-up

```
(define c (make-custodian))
(parameterize ((current-custodian c))
  ...)
```

eval

```
(custodian-shutdown-all c)
```

eval

Custodians

Resource limits

```
(define (run-away)
  (cons 1 (run-away)))

(custodian-limit-memory c 1000000 c)

(parameterize ((current-custodian c))
  ...
  (thread run-away))
```

eval

Etc.

- Security Guards

  Resource access control

- Namespaces

  Global bindings

- Will Executors

  Timing of finalizations

- Inspectors

  Debugging access

- **Motivation and Approach**
- **PLT Scheme as an Operating System**
- **Putting the Pieces to Work**
  - SchemeEsq, a mini DrScheme [ICFP99]

GUI - Frame

```
(define frame
  (new frame%
    [label "SchemeEsq"]
    [width 400] [height 175]))

(send frame show #t)
```

eval

GUI - Reset Button

```
(new button%
  [label "Reset"]
  [parent frame]
  [callback (lambda (b e) (reset-program))])
```

eval

GUI - Interaction Area

```
(define repl-display-canvas
  (new editor-canvas%
    [parent frame]))
```

eval

GUI - Interaction Buffer

```
(define esq-text%
  (class text% ... (evaluate str) ...))

(define repl-editor (new esq-text%))
(send repl-display-canvas set-editor repl-editor)
```

eval

Evaluator

```
(define (evaluate expr-str)
  (thread
    (lambda ()
      (print (eval (read (open-input-string expr-str))))
      (newline)
      (send repl-editor new-prompt))))
```

eval

Evaluator Output

```
(define user-output-port
  (make-custom-output-port ... repl-editor ...))

(define (evaluate expr-str)
  (parameterize ((current-output-port user-output-port))
    (thread
      (lambda ()
        ...))))
```

eval

Evaluating GUIs

```
(define user-eventspace (make-eventspace))

(define (evaluate expr-str)
  (parameterize ((current-output-port user-output-port)
                 (current-eventspace user-eventspace))
    (thread
      (lambda ()
        ...)))
```

eval

Custodian for Evaluation

```
(define user-custodian (make-custodian))

(define user-eventspace
  (parameterize ((current-custodian user-custodian))
    (make-eventspace)))

(define (evaluate expr-str)
  (parameterize ((current-output-port user-output-port)
                 (current-eventspace user-eventspace)
                 (current-custodian user-custodian))
    (thread
      (lambda ()
        ...))))
```

eval

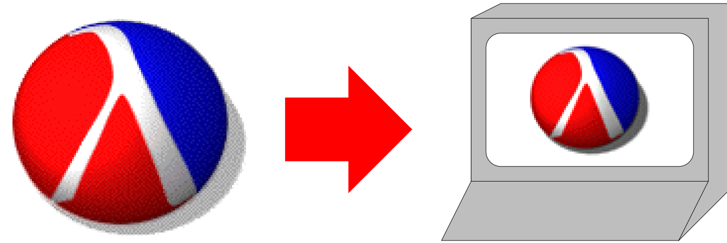
Reset Evaluation

```
(define (reset-program)
  (custodian-shutdown-all user-custodian)

  (set! user-custodian (make-custodian))
  (parameterize ((current-custodian user-custodian))
    (set! user-eventspace (make-eventspace)))
  (send repl-editor reset))
```

eval

Conclusion



- Programmers need OS-like constructs in languages
  - concurrency
  - adjust run-time environment
  - easy termination
- Multiple language constructs for "process"
  - programmer can mix and match to balance isolation and cooperation