

Processes without Partitions



Matthew Flatt

University of Utah

Adam Wick

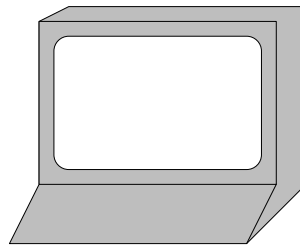
University of Utah

Robert Bruce Findler

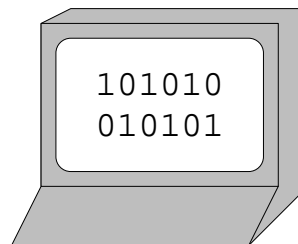
University of Chicago

This talk has been modified from its original version. It has been edited for content and formatted to fit your screen.

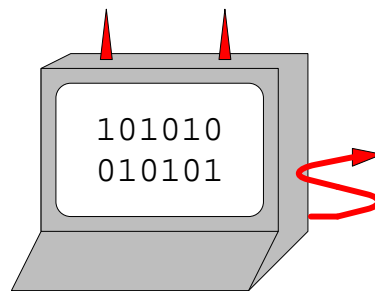
Machines and Processes



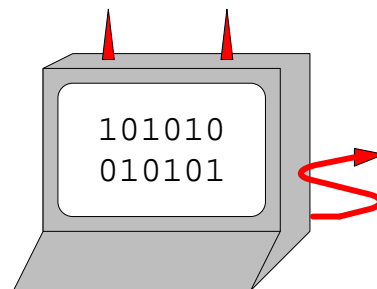
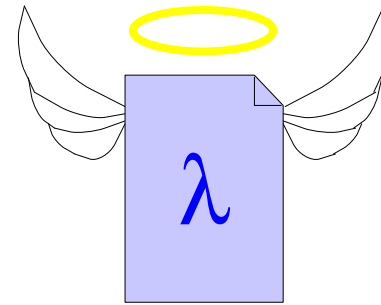
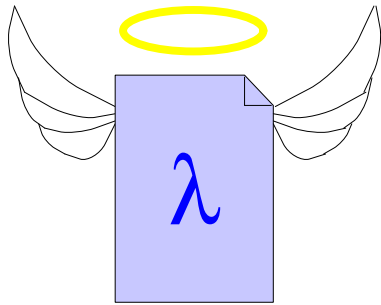
Machines and Processes



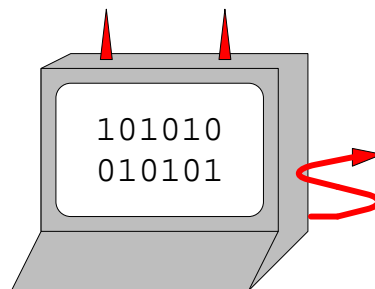
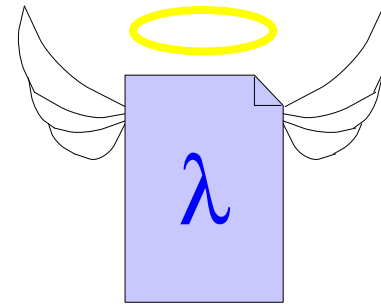
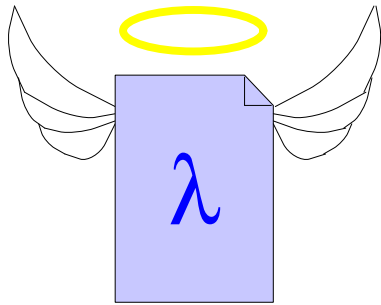
Machines and Processes



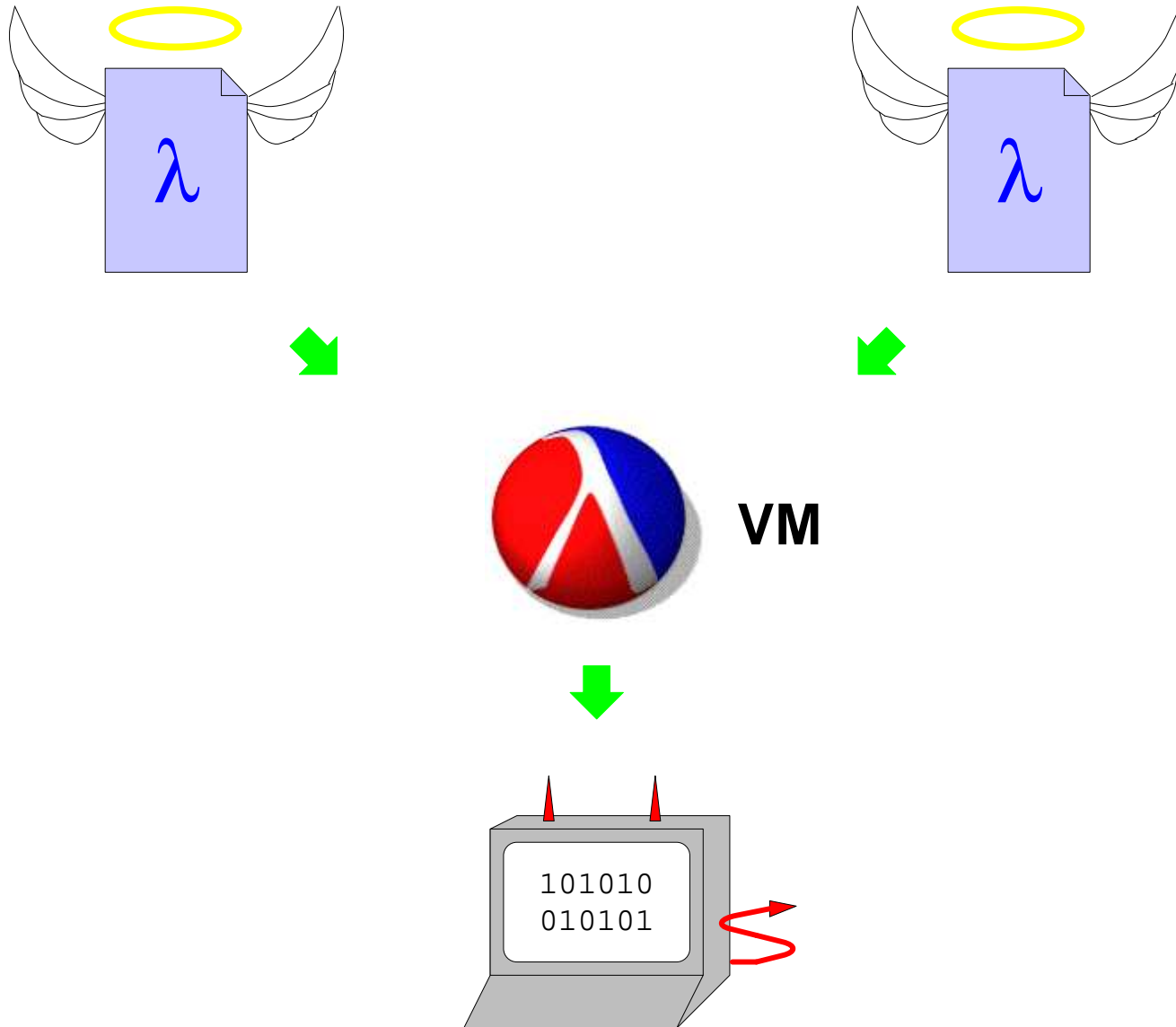
Machines and Processes



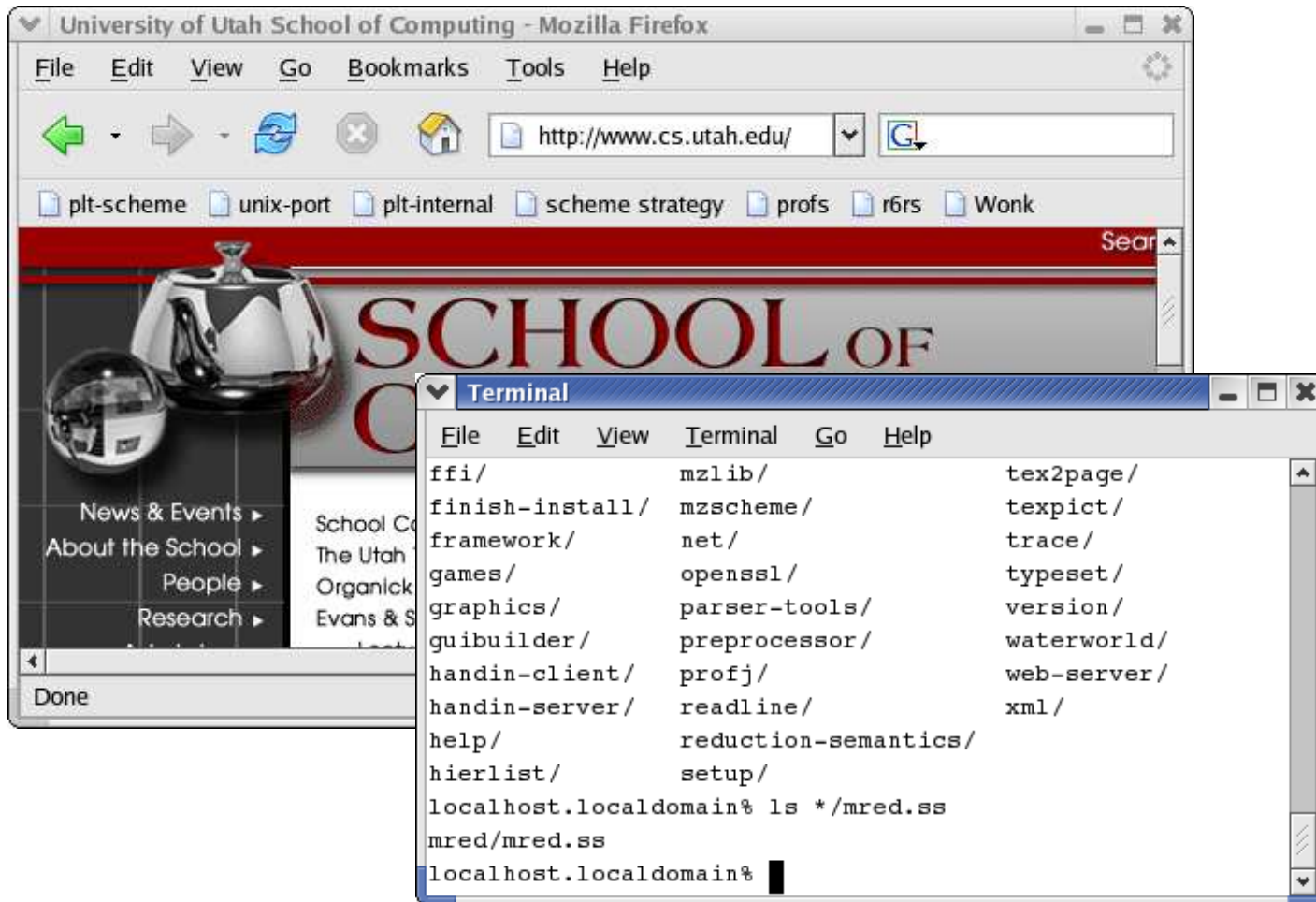
Machines and Processes



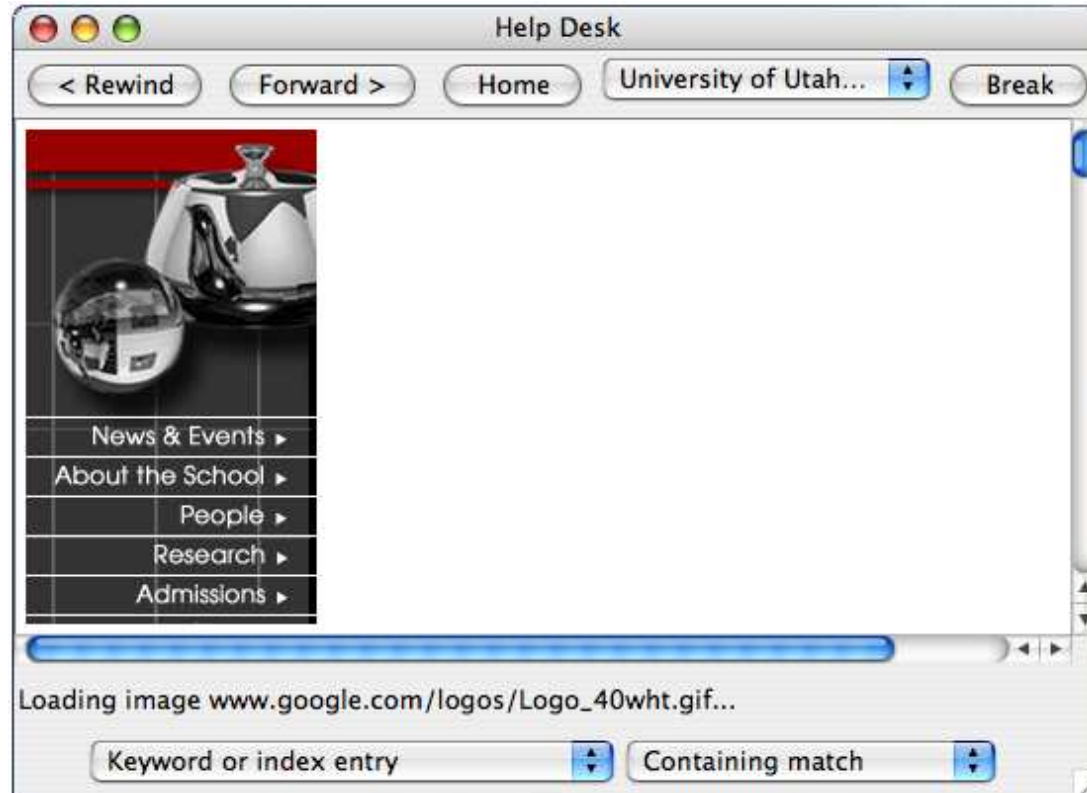
Machines and Processes



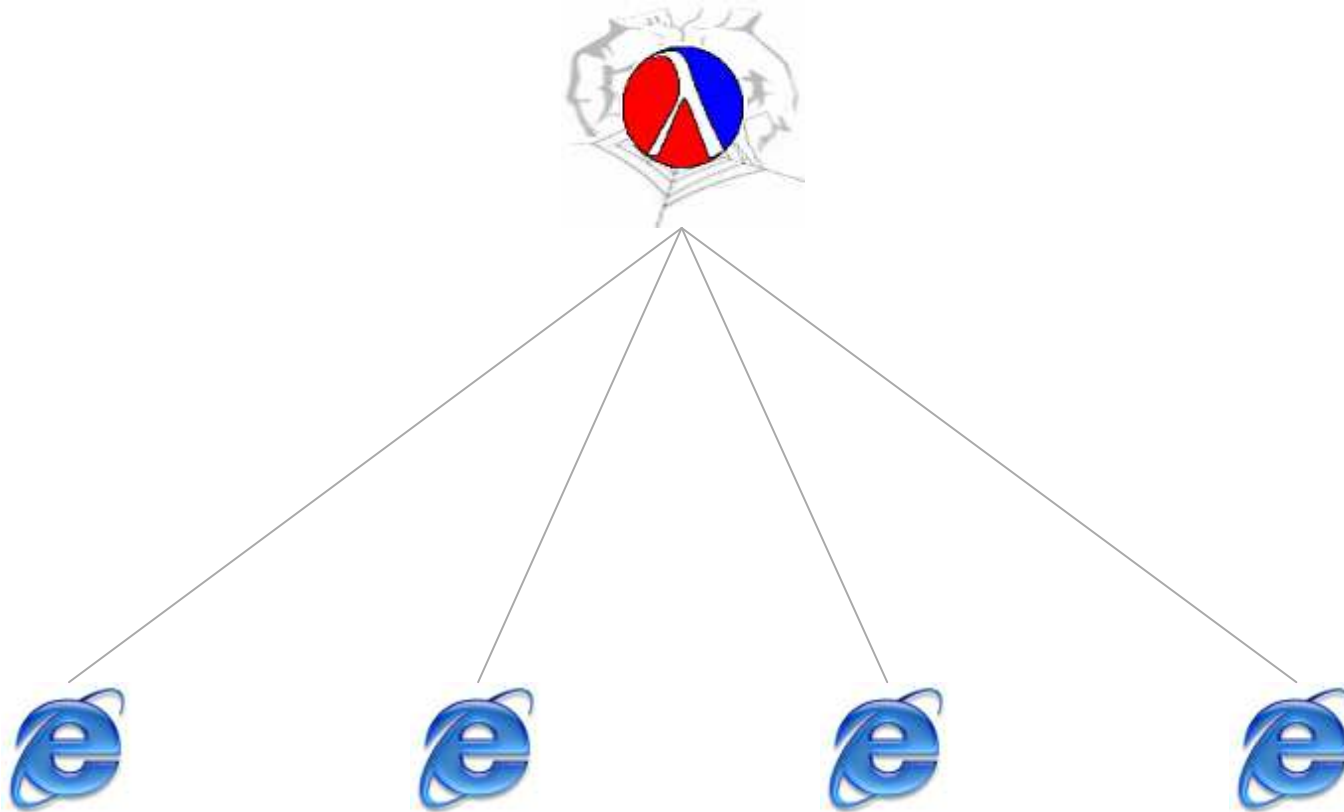
Process Examples



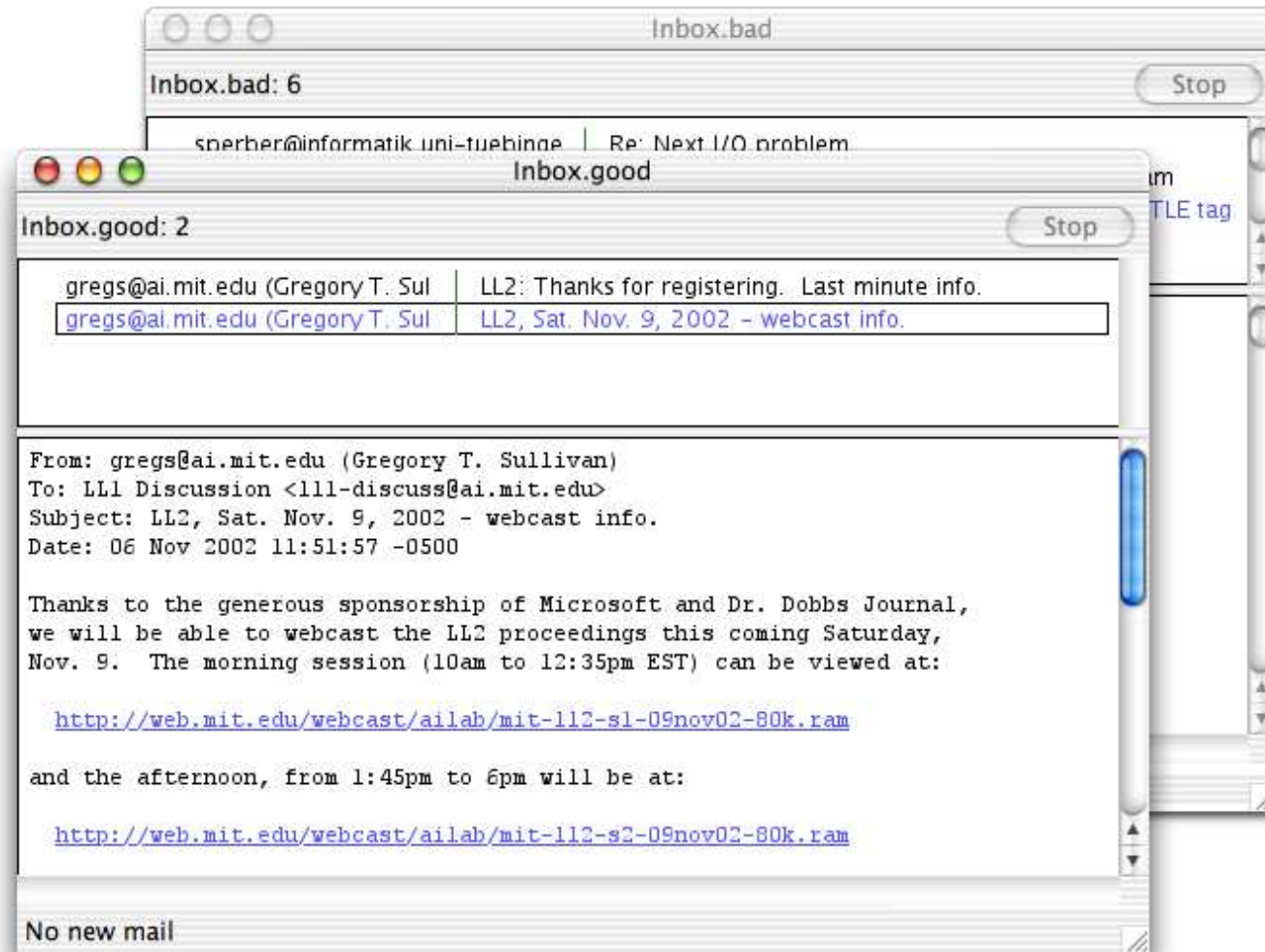
Process Examples



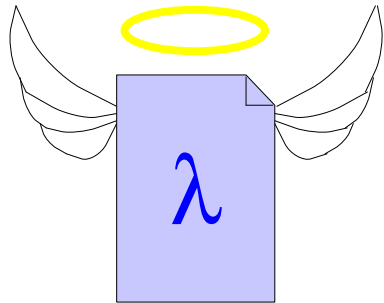
Process Examples



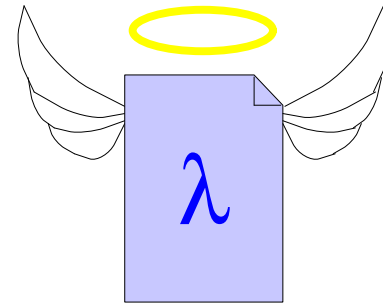
Process Examples



Process Examples



DrScheme



user's program



Run DrScheme

Languages with Termination

Pilot [Redell80]

SPIN [Bershad95]

JKernel [Hawblitzel98]

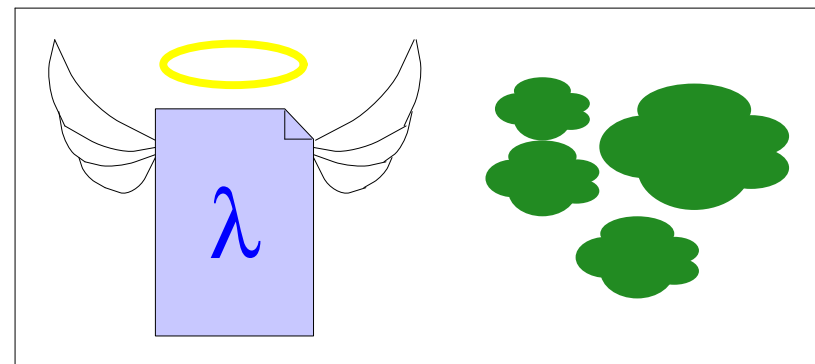
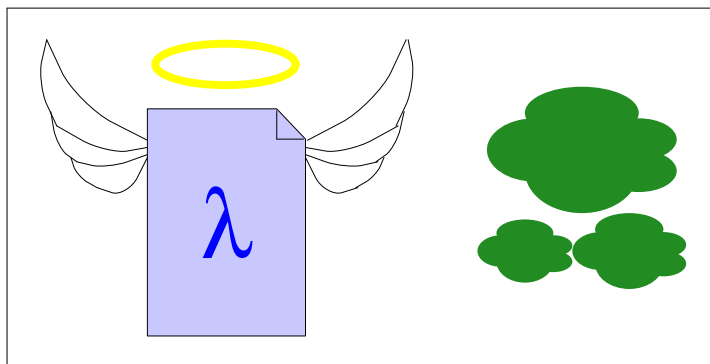
Alta [Tullman99]

KaffeOS [Back00]

JSR-121 [Soper03]

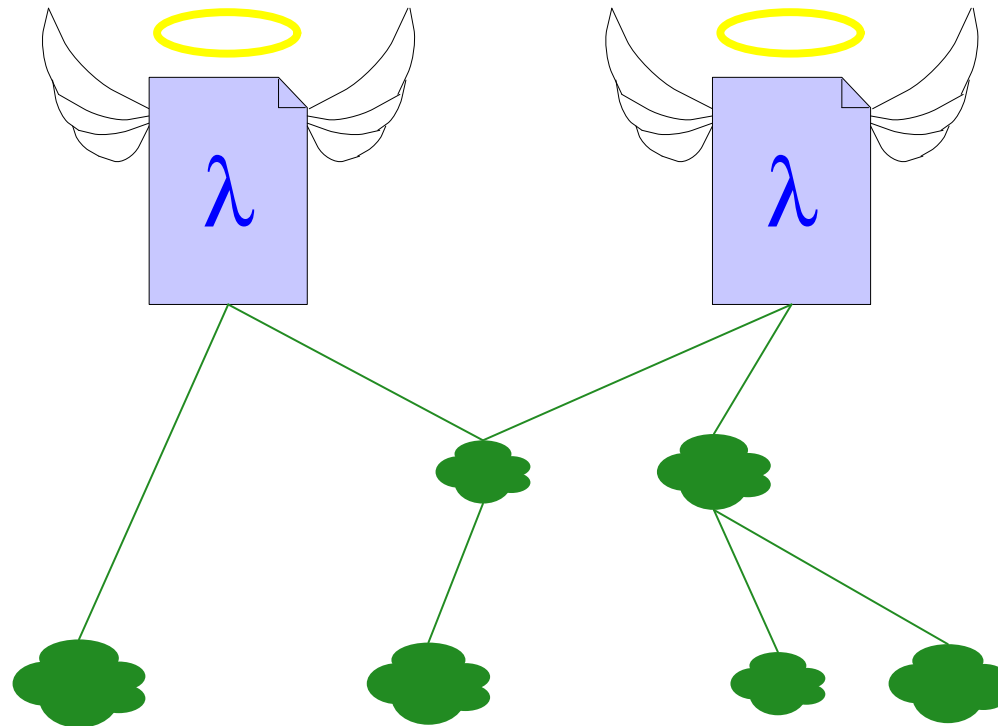
.NET application domains

...



Languages with Termination

PLT Scheme



➤ **Motivation and Approach**

➤ **Processes in PLT Scheme**

- Threads
- Parameters
- Eventspaces
- Custodians

➤ **Memory Accounting**

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
```

Etc.

- Security Guards

Resource access control

- Namespaces

Global bindings

- Will Executors

Timing of finalizations

- Inspectors

Debugging access

Building a Programming Environment

[SchemeEsq](#), a mini DrScheme [ICFP 99]

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-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

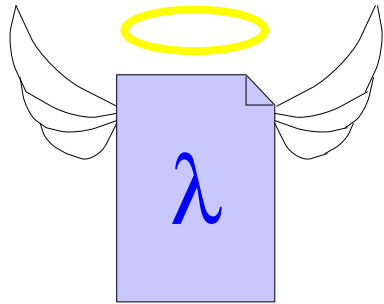
```
(define (reset-program)
  (custodian-shutdown-all user-custodian)

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

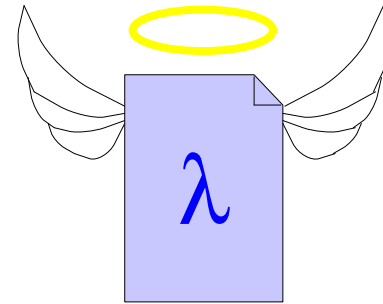
eval

- **Motivation and Approach**
- **Processes in PLT Scheme**
- **Memory Accounting**
 - Without partitions [ISMM 04]

Resource Consumption



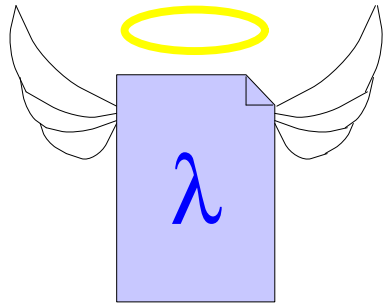
DrScheme



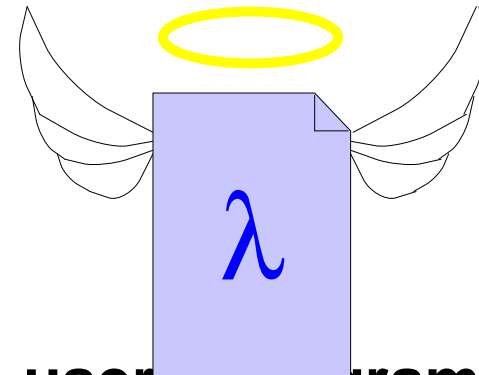
user's program



Resource Consumption



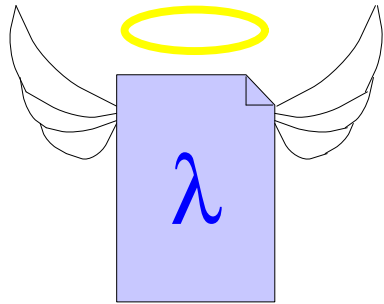
DrScheme



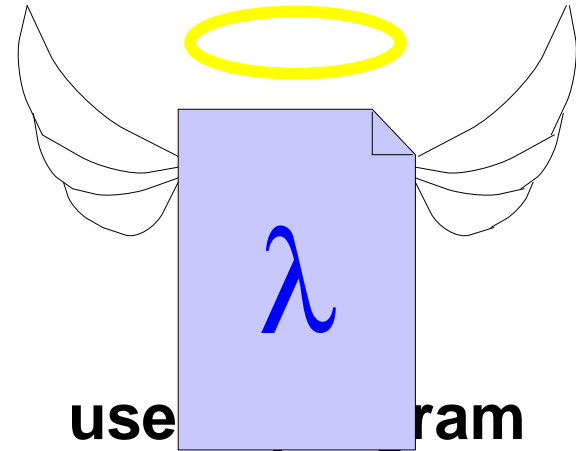
user's program



Resource Consumption



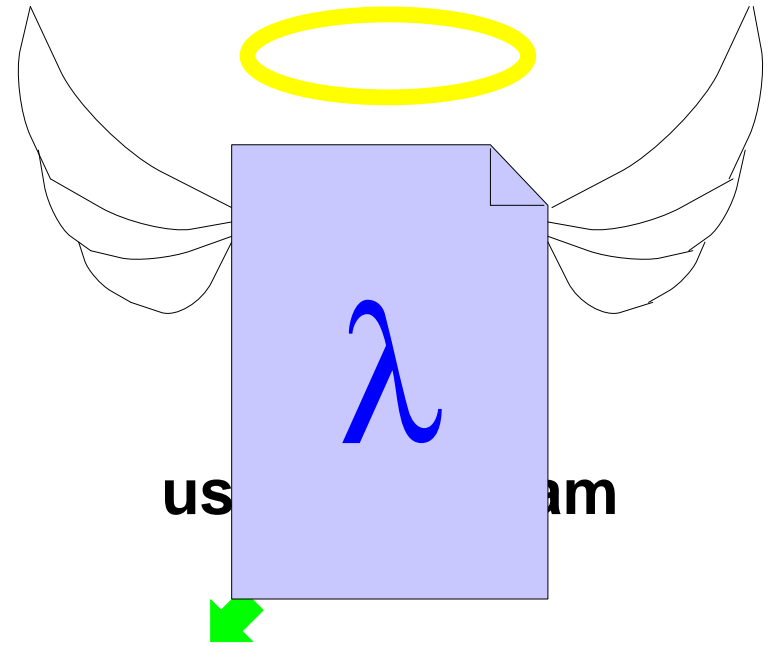
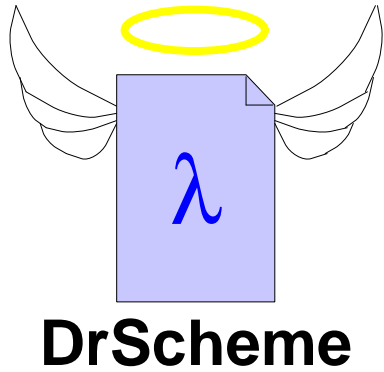
DrScheme



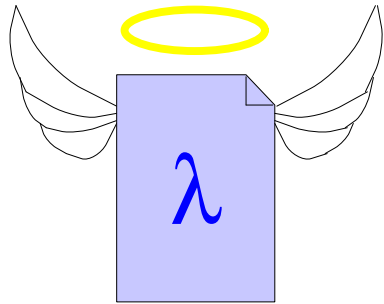
use program



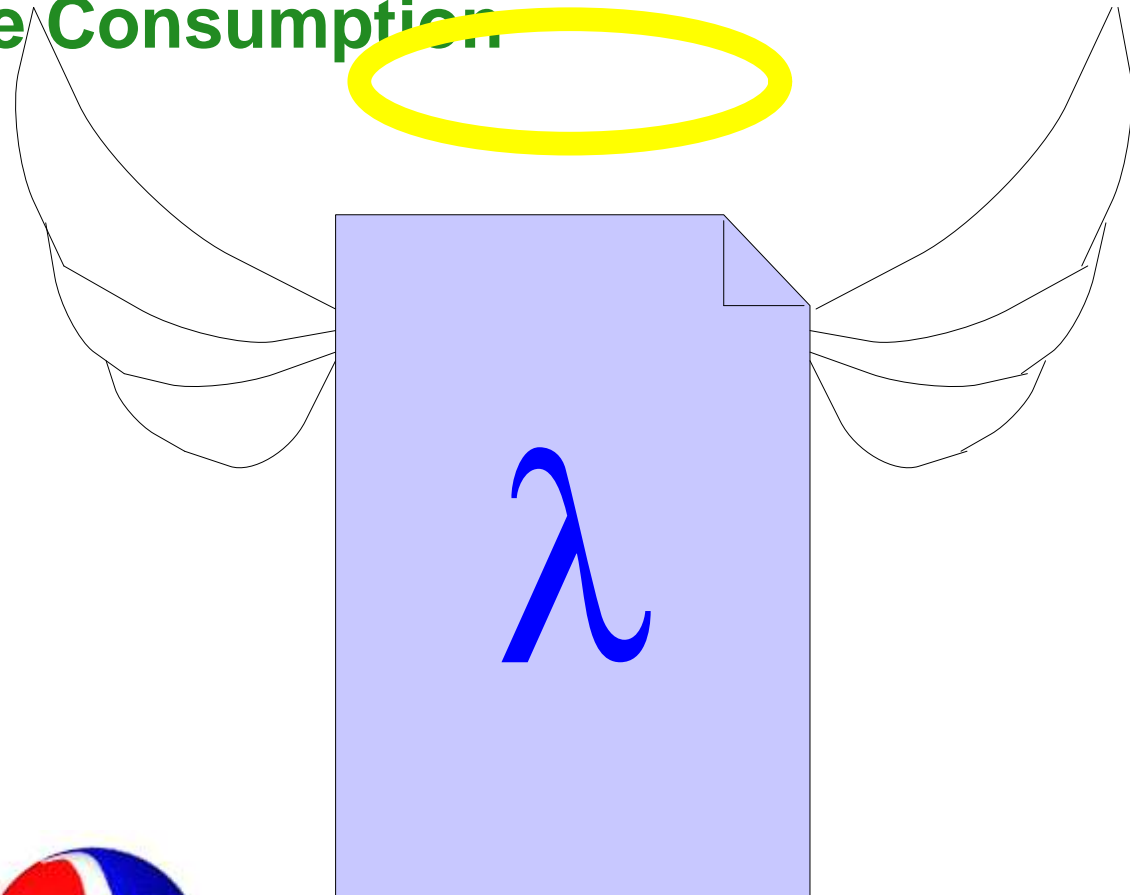
Resource Consumption



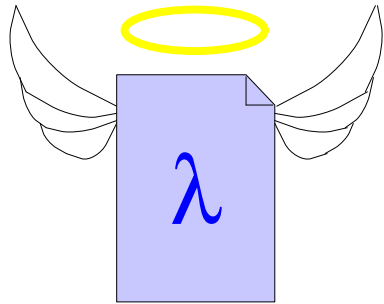
Resource Consumption



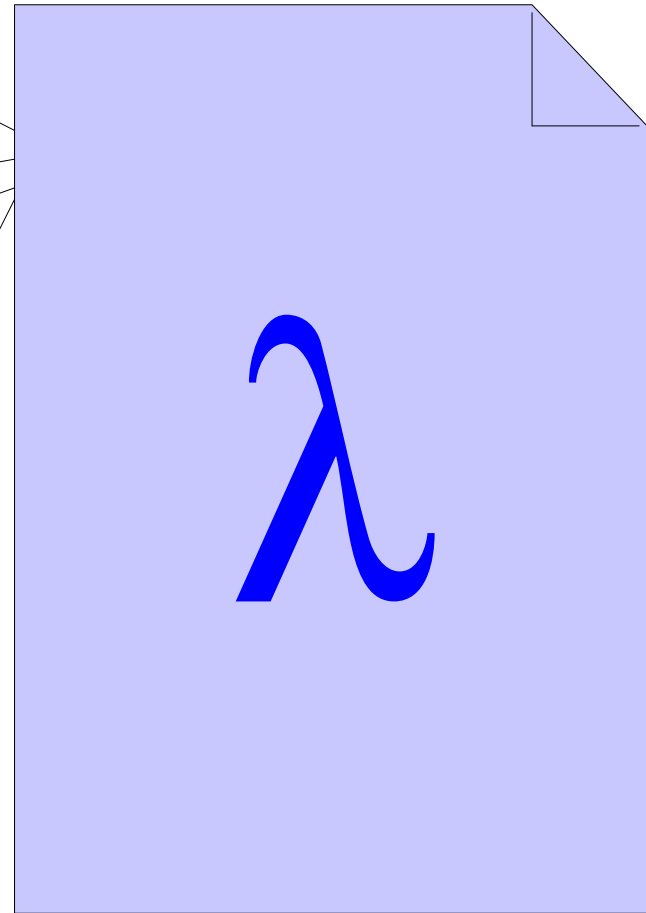
DrScheme



Resource Consumption

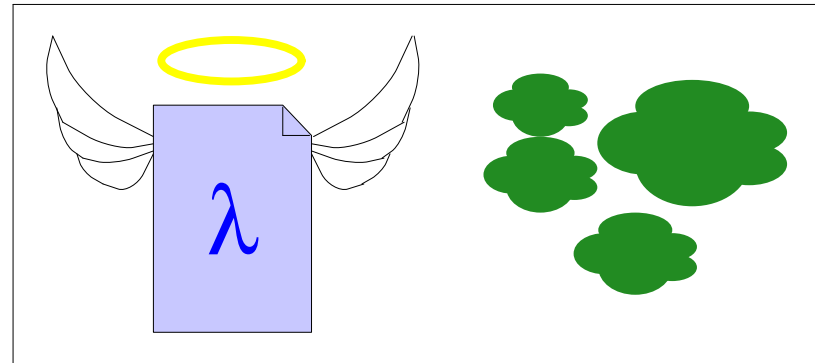
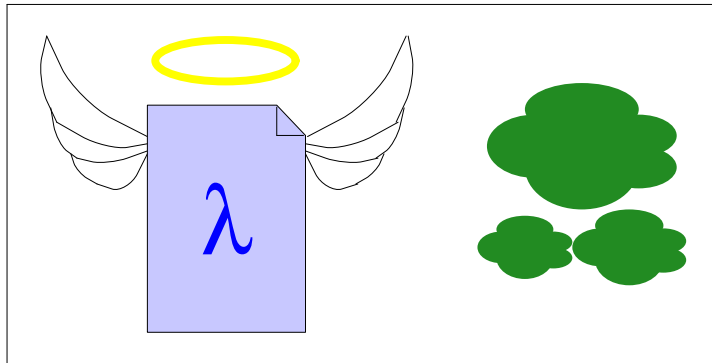


DrScheme



Resource Accounting

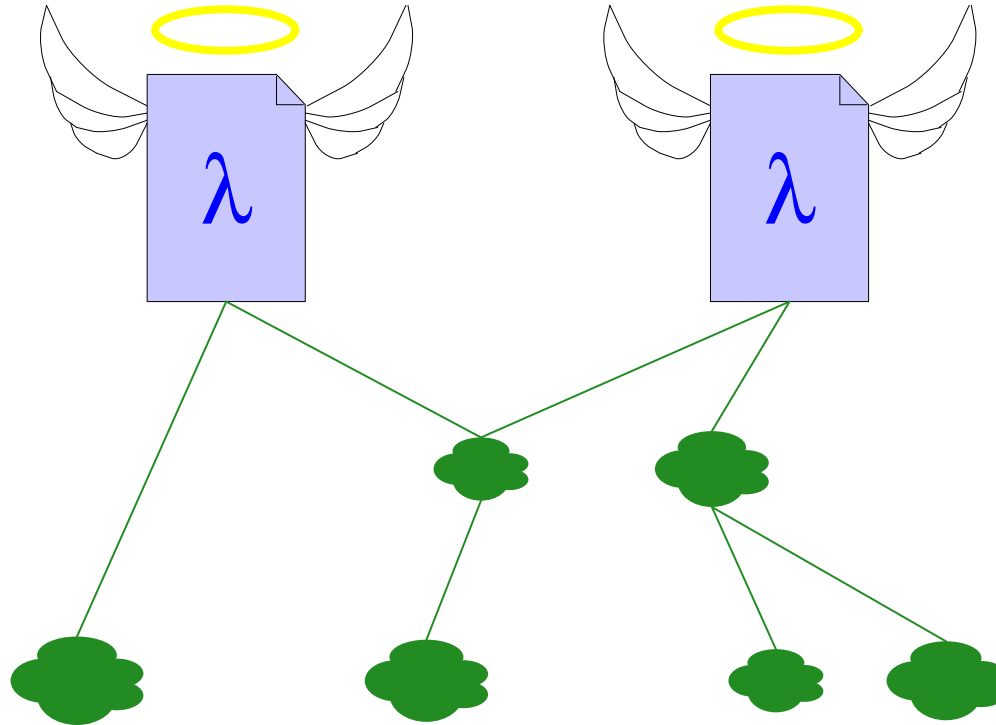
- **Conventional OS:** process memory use = size of partition



- Accounting is easy
- Trading data is difficult

Resource Accounting

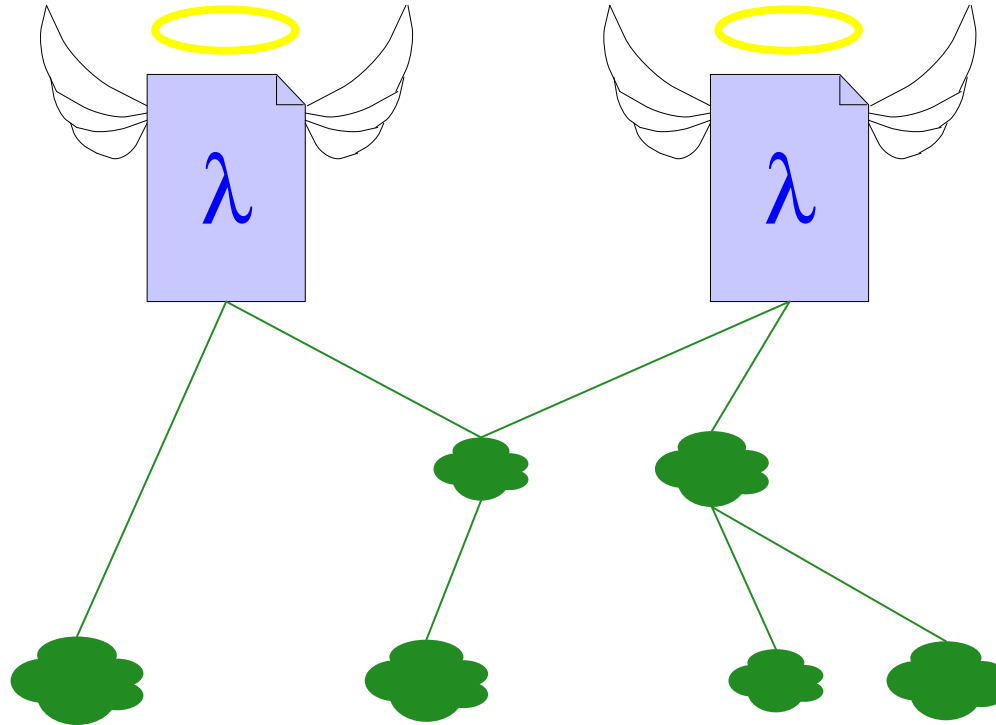
- **Language as OS**: process memory use = size of owned data



- Trading data is easy
- Accounting *appears* difficult: sharing, real-time tracking

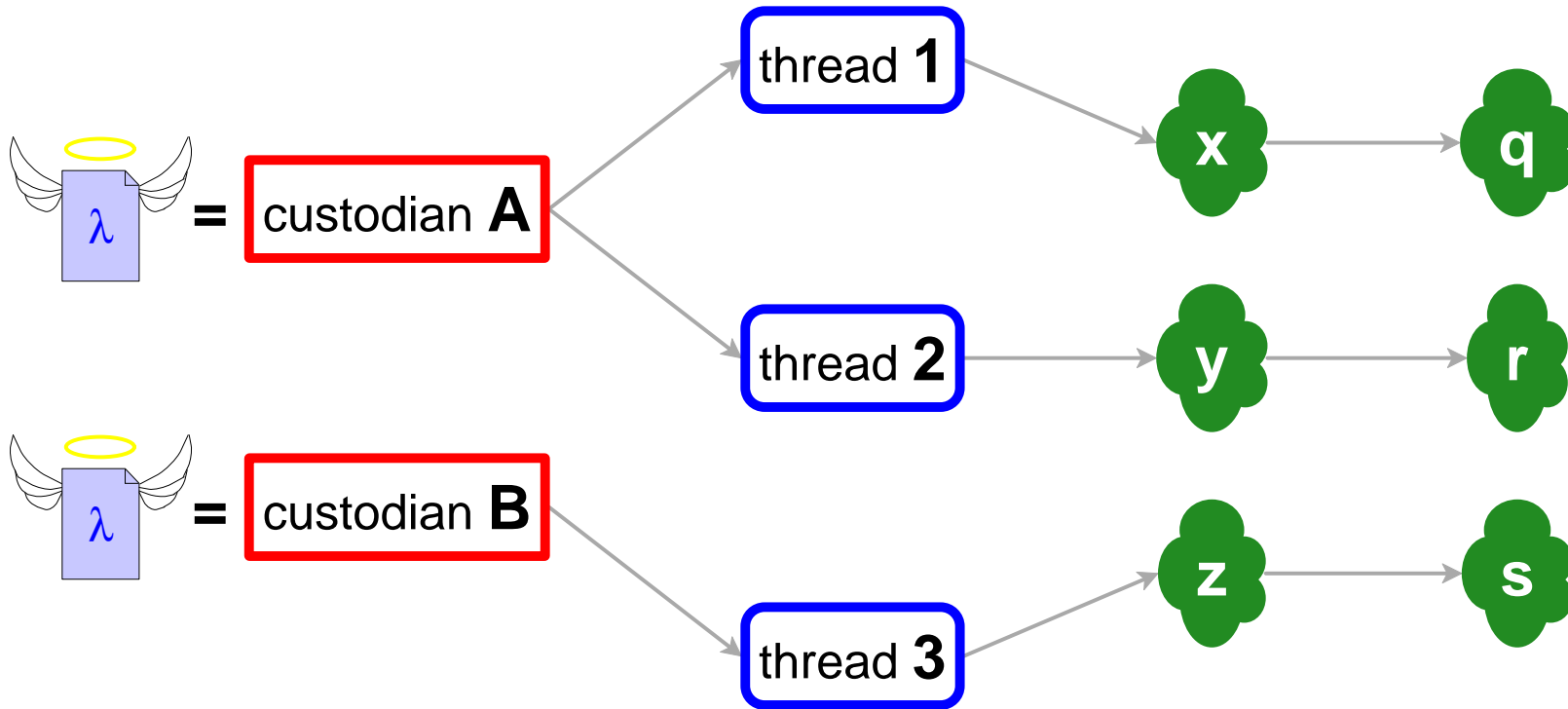
Resource Accounting

Our strategy: compute accounting charges during GC

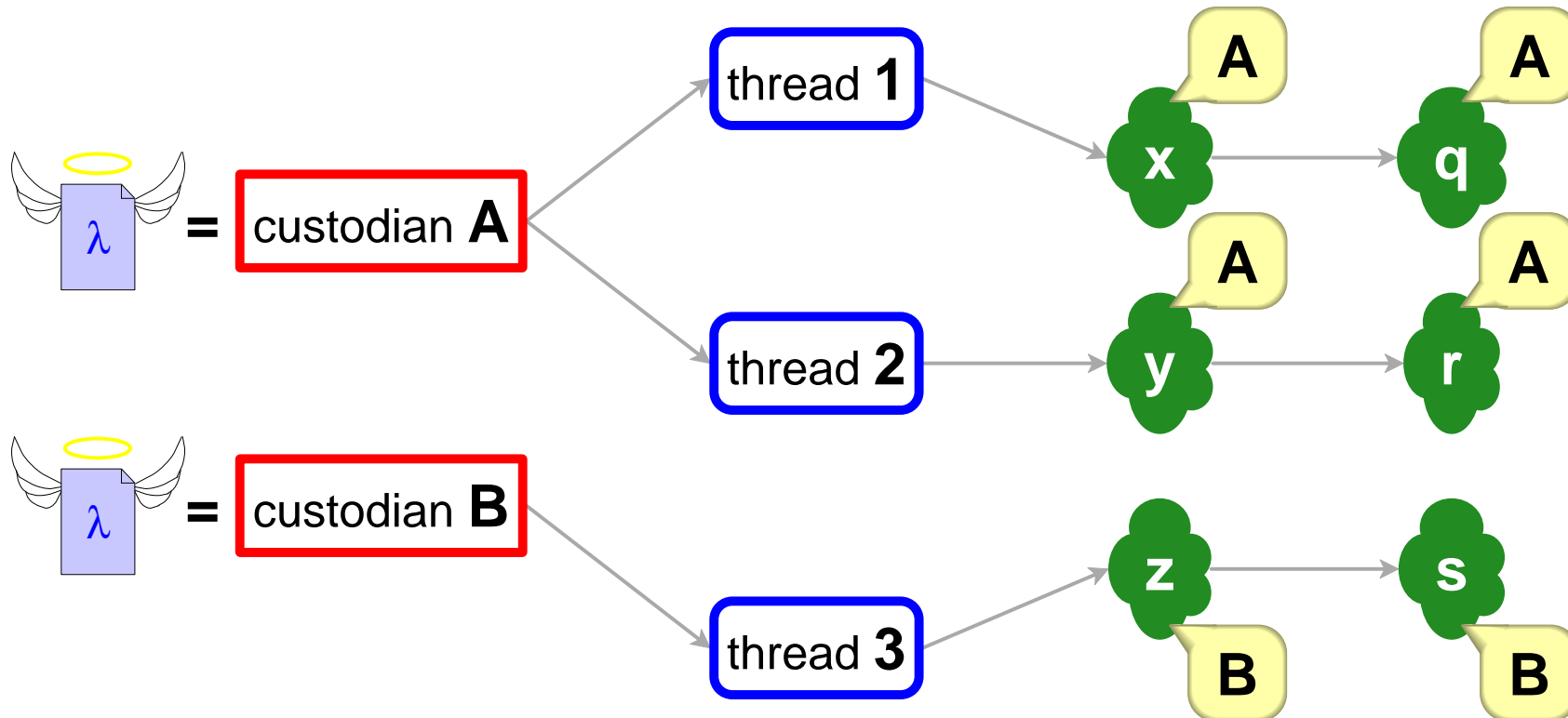


See also [Price03]

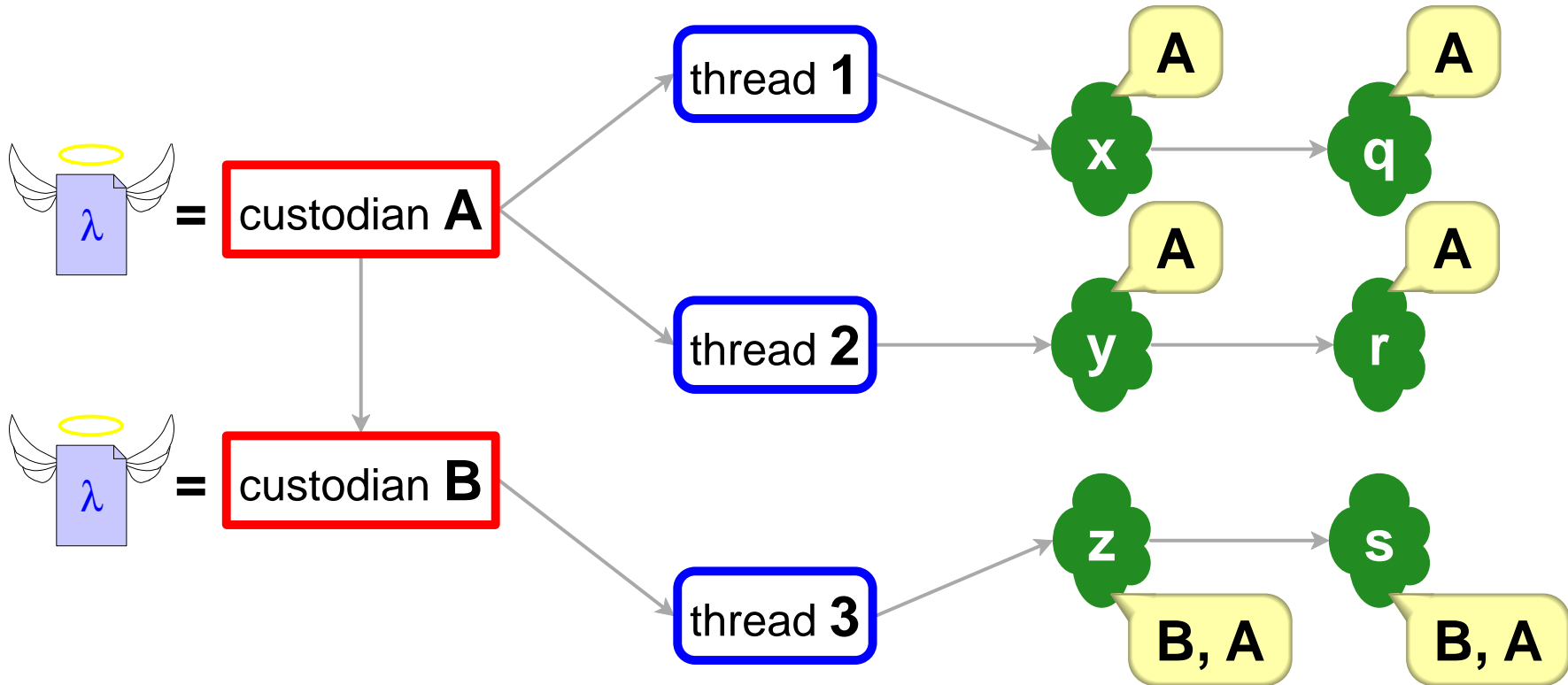
Basic Accounting



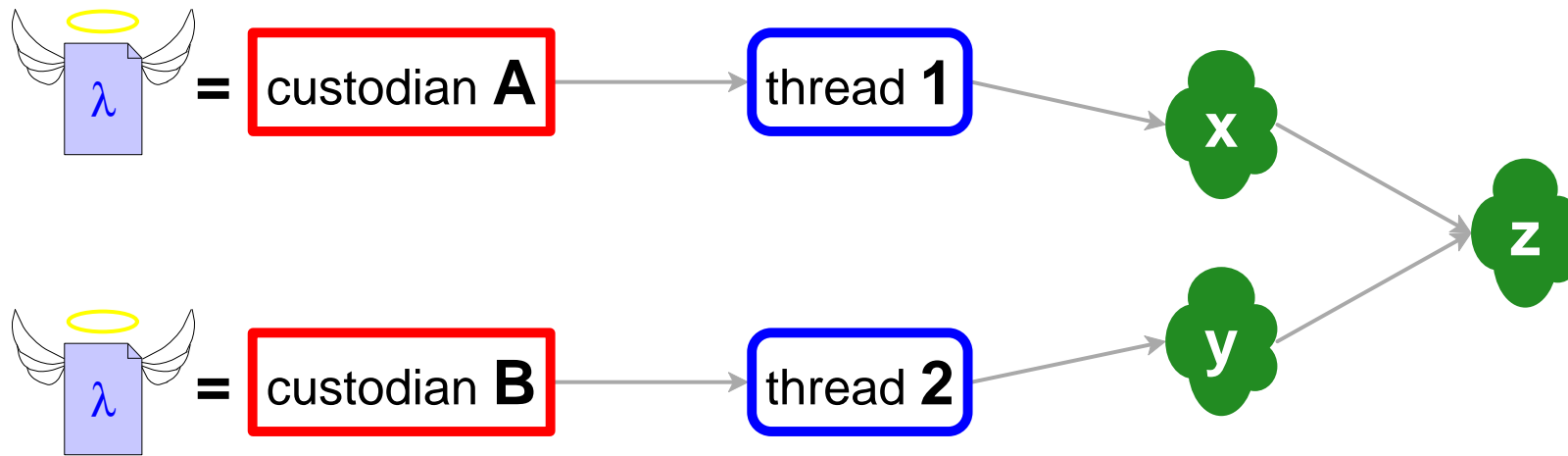
Basic Accounting



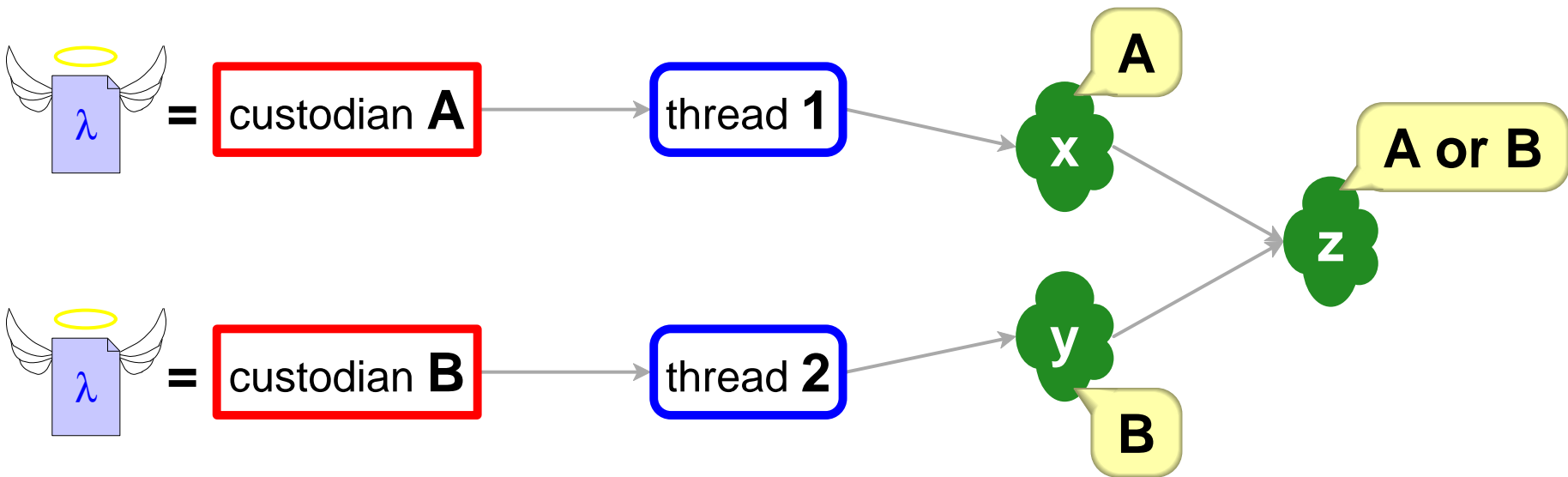
Basic Accounting



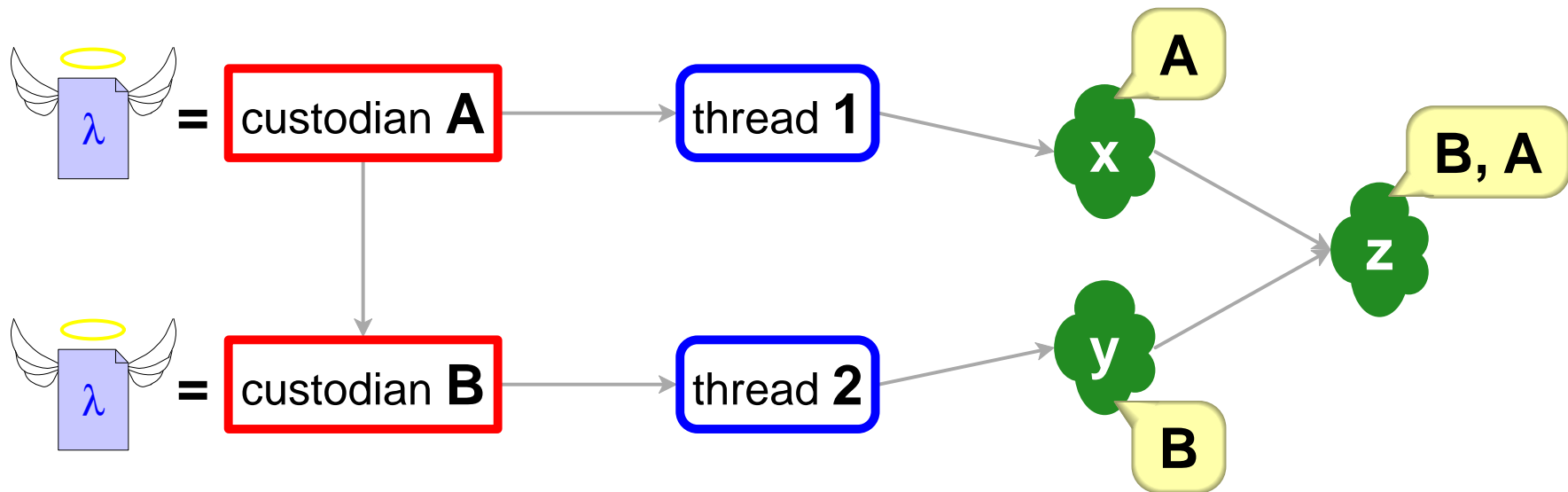
Sharing



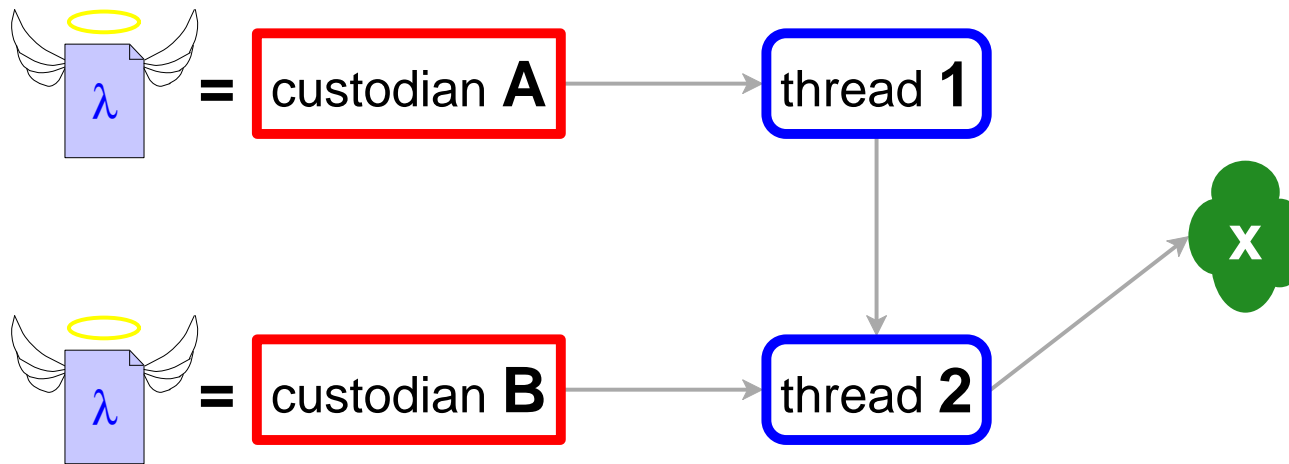
Sharing



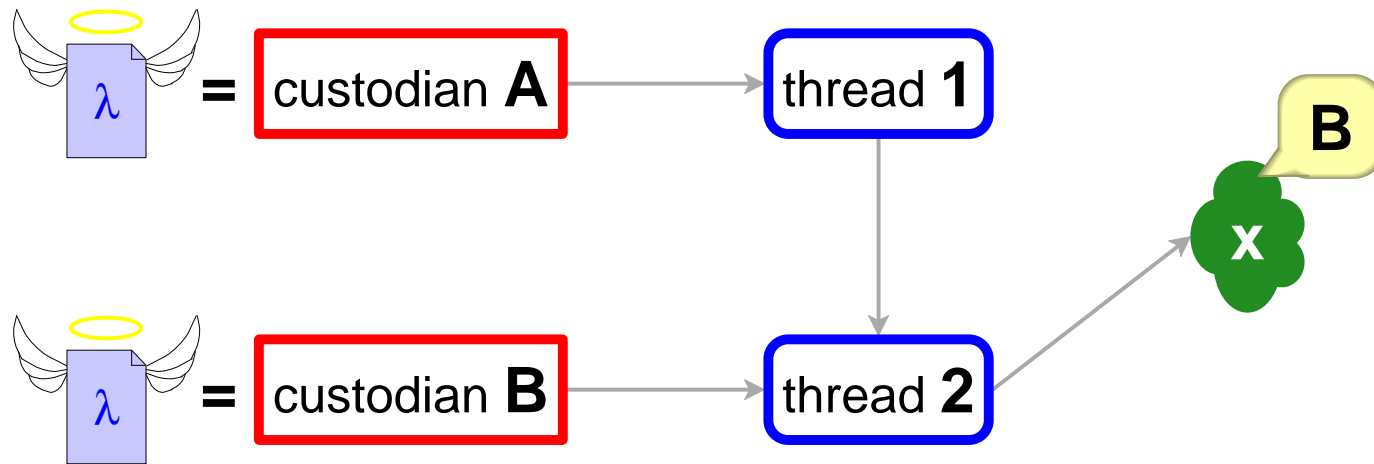
Sharing: Charge the Child



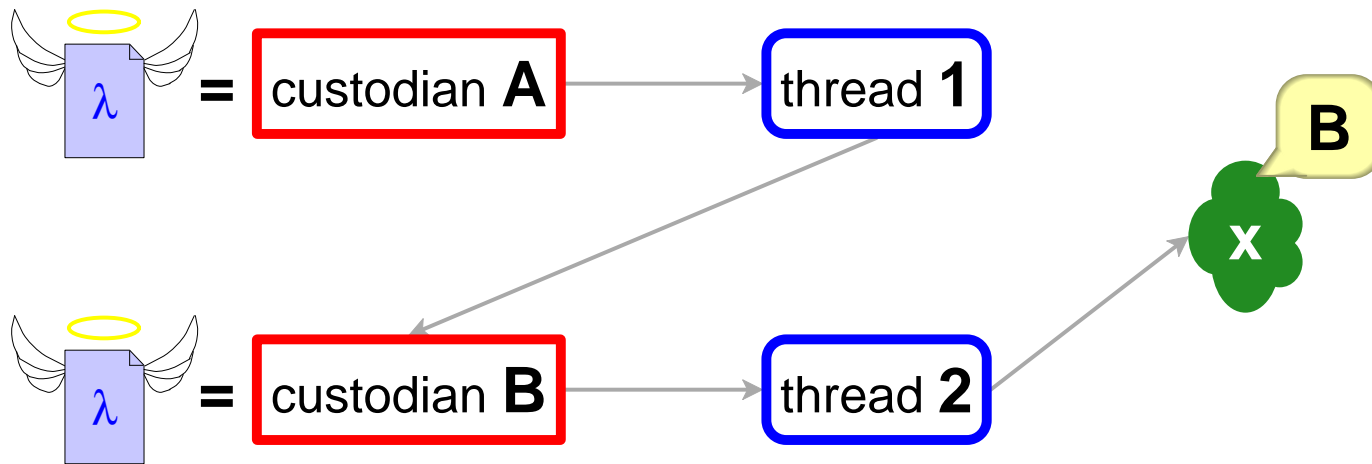
Threads, Custodians, and Weak References



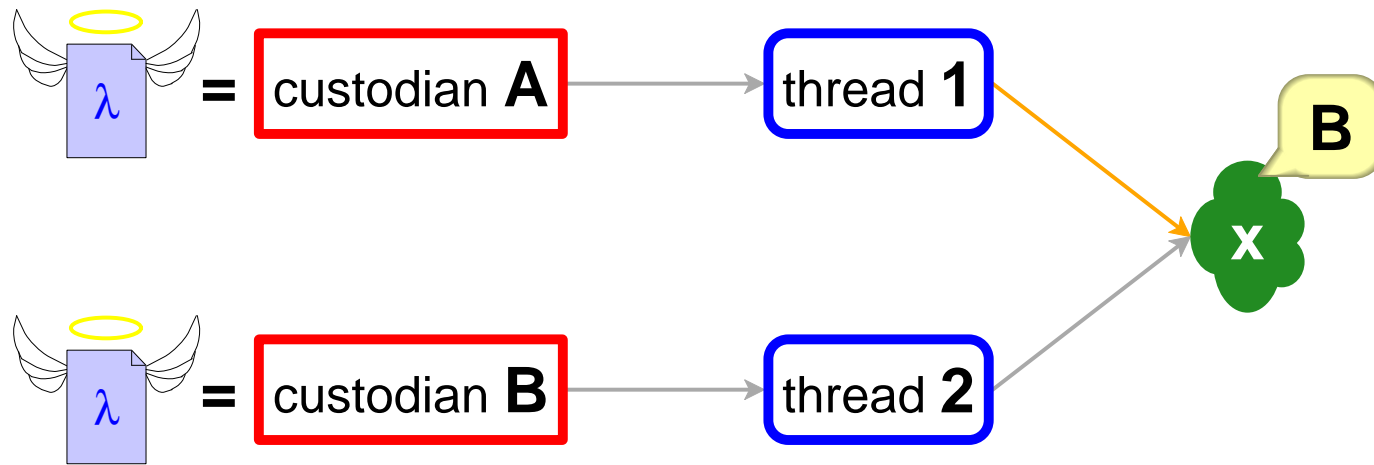
Threads, Custodians, and Weak References



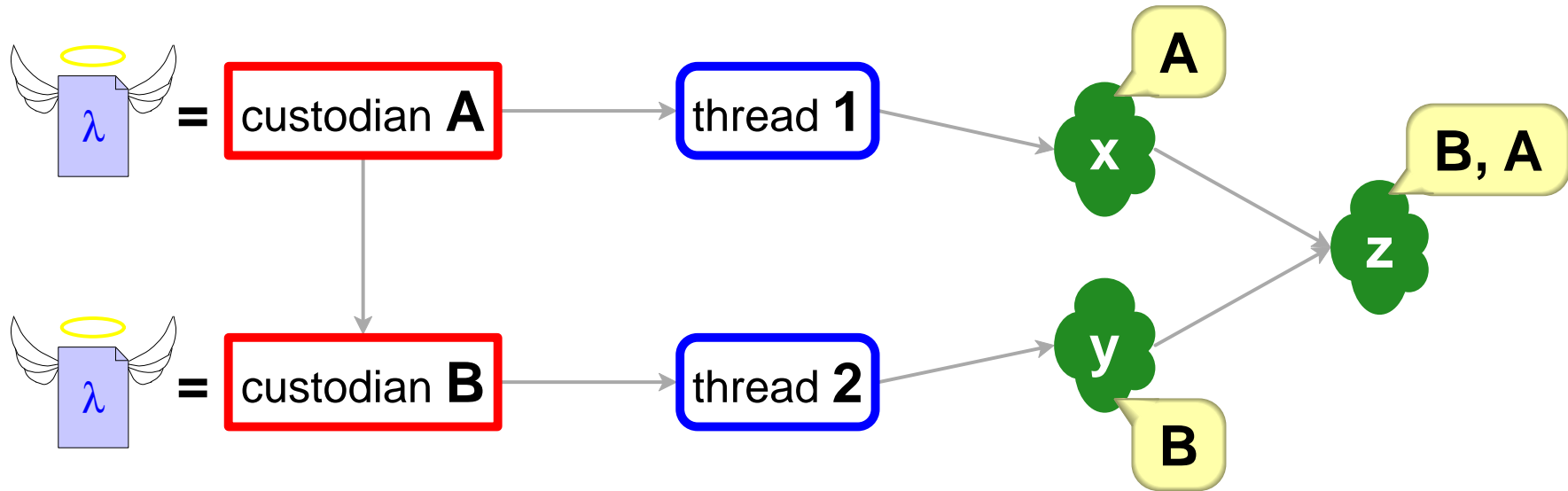
Threads, Custodians, and Weak References



Threads, Custodians, and Weak References

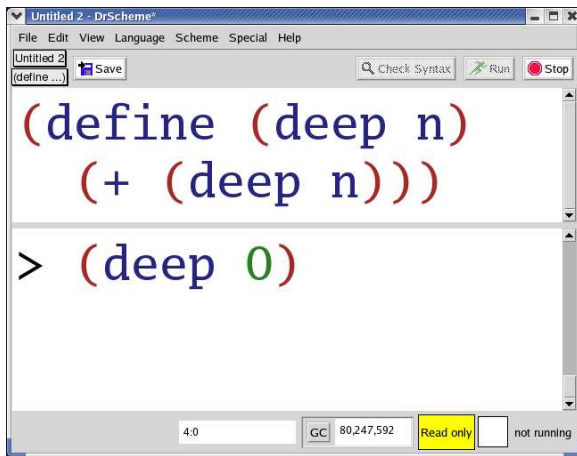


Why Charge the Child?



- Parent is responsible for children
- Parent may allocate for children
 - GUI objects
 - File descriptors
 - ...

Initial Experience: DrScheme

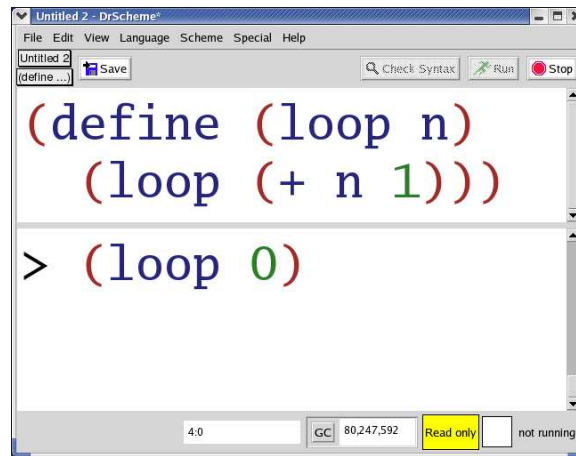


```
(define (deep n)
  (+ (deep n)))

> (deep 0)
```

4.0 GC 80,247,592 Read only not running

Bad Loop

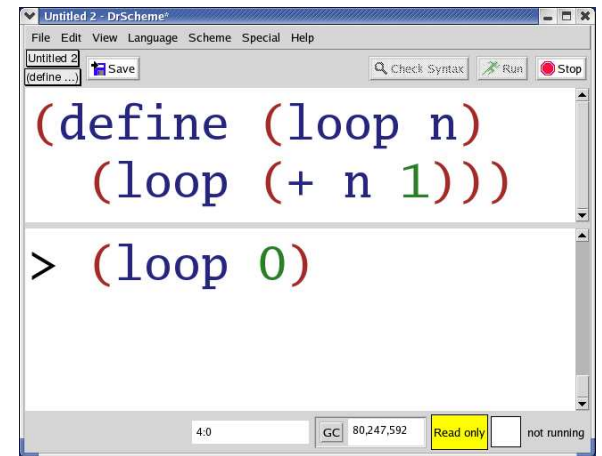


```
(define (loop n)
  (loop (+ n 1)))

> (loop 0)
```

4.0 GC 80,247,592 Read only not running

Normal



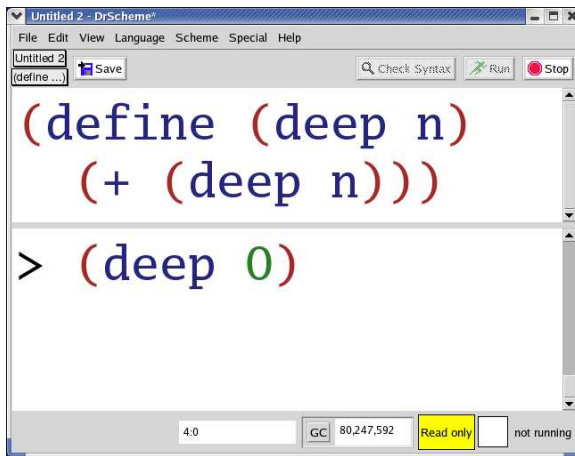
```
(define (loop n)
  (loop (+ n 1)))

> (loop 0)
```

4.0 GC 80,247,592 Read only not running

Normal

Initial Experience: DrScheme

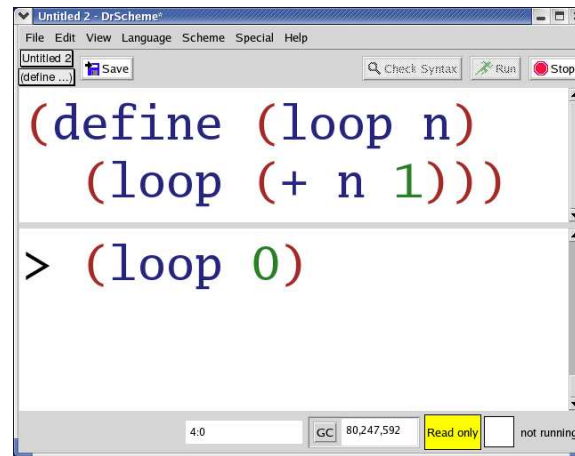


DrScheme window showing a bad loop. The code defines a recursive function `(define (deep n) (+ (deep n)))` and then calls `(deep 0)`. The status bar at the bottom indicates the program is not running.

```
(define (deep n)
  (+ (deep n)))

> (deep 0)
```

Bad Loop

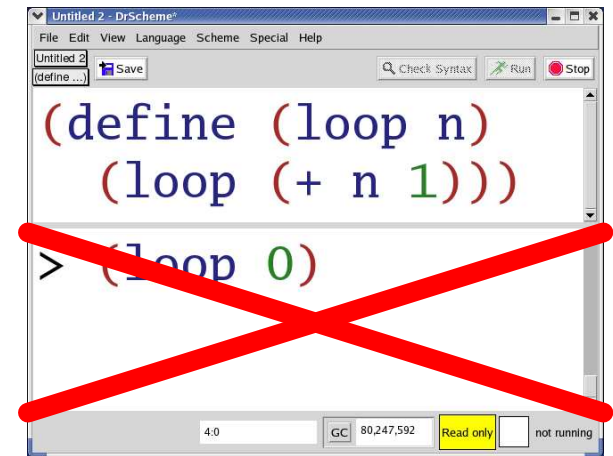


DrScheme window showing normal execution. The code defines a recursive function `(define (loop n) (loop (+ n 1)))` and then calls `(loop 0)`. The status bar at the bottom indicates the program is not running.

```
(define (loop n)
  (loop (+ n 1)))

> (loop 0)
```

Normal



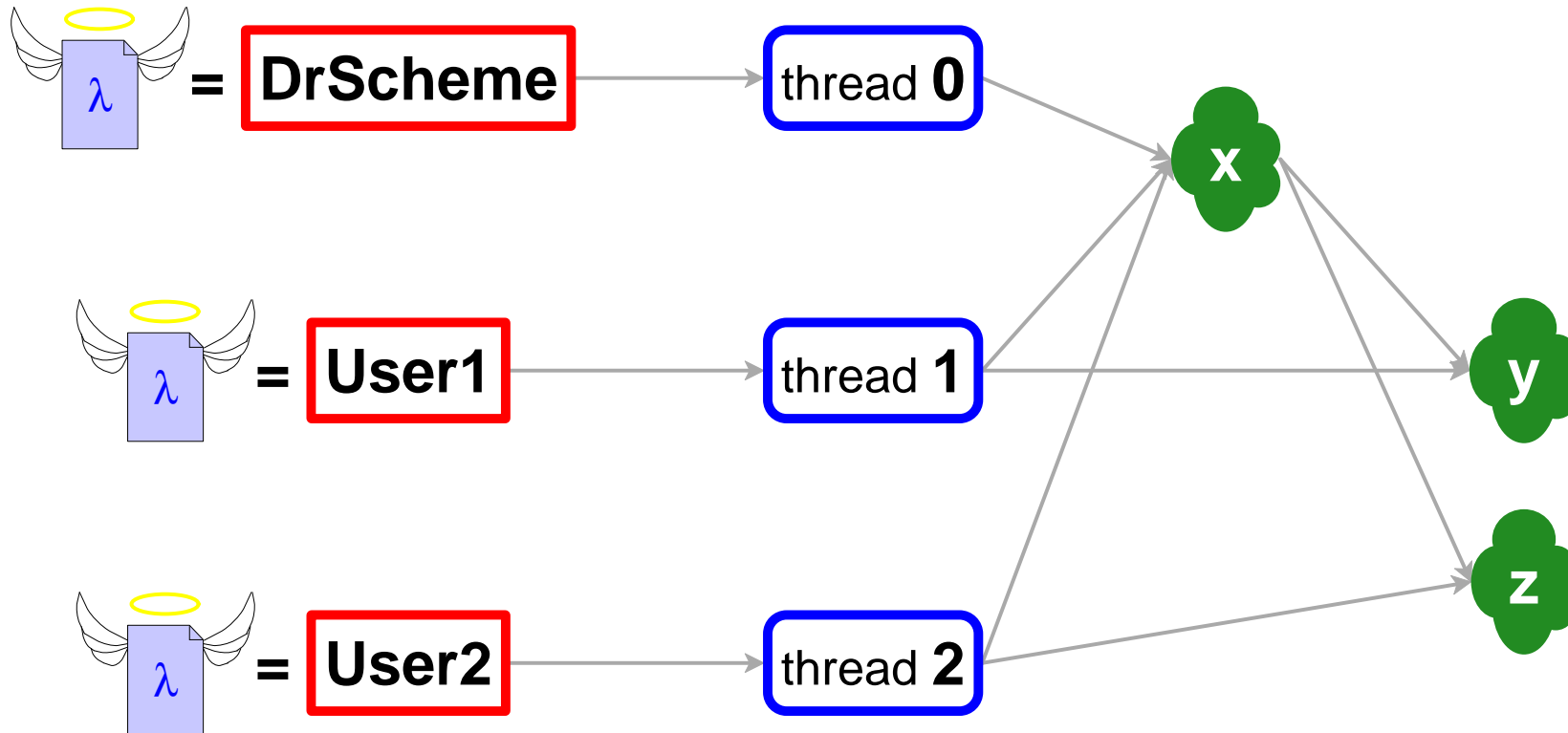
DrScheme window showing a shut down state. The code defines a recursive function `(define (loop n) (loop (+ n 1)))` and then calls `(loop 0)`. The entire window is crossed out with a large red X, indicating that the program has been terminated.

```
(define (loop n)
  (loop (+ n 1)))

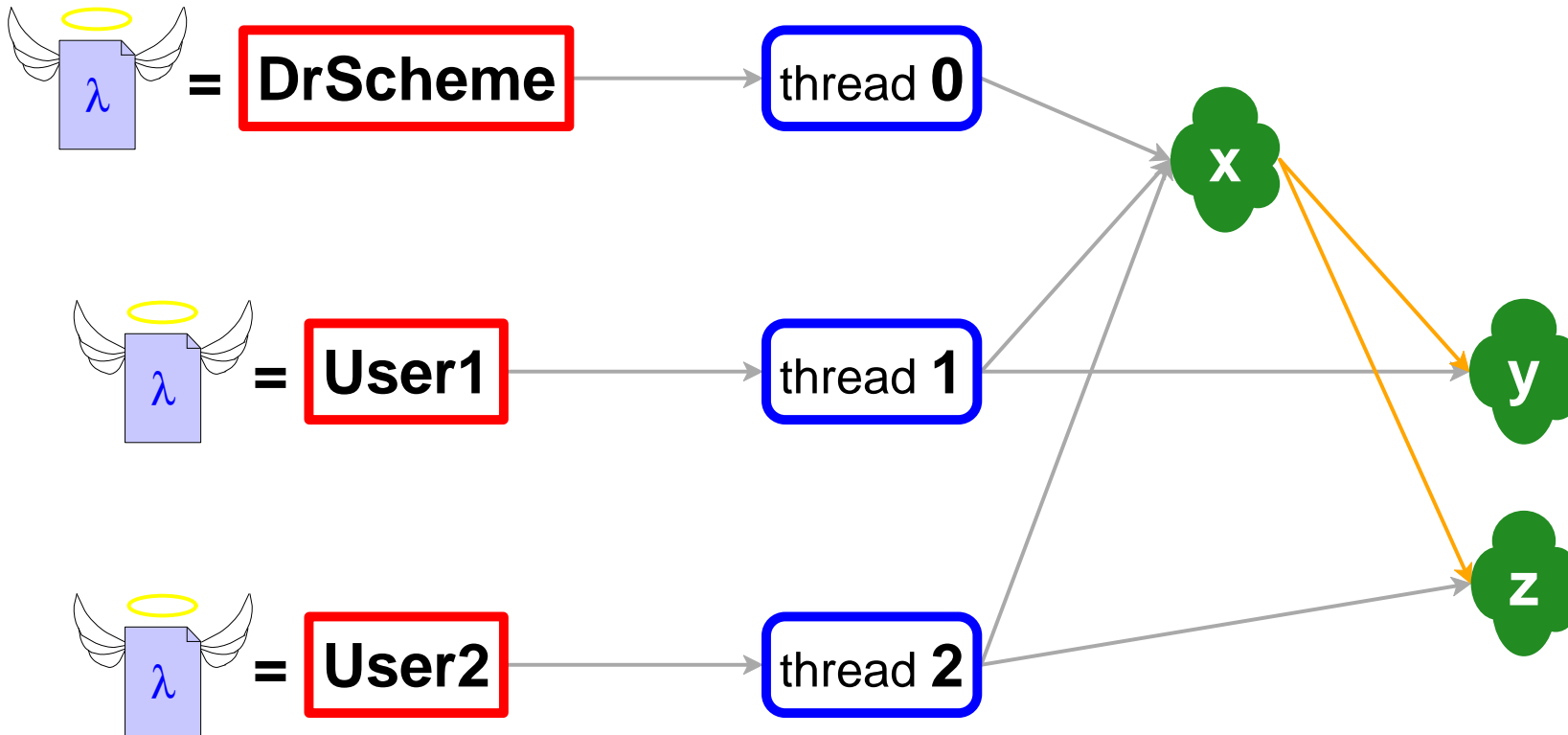
> (loop 0)
```

Shut Down

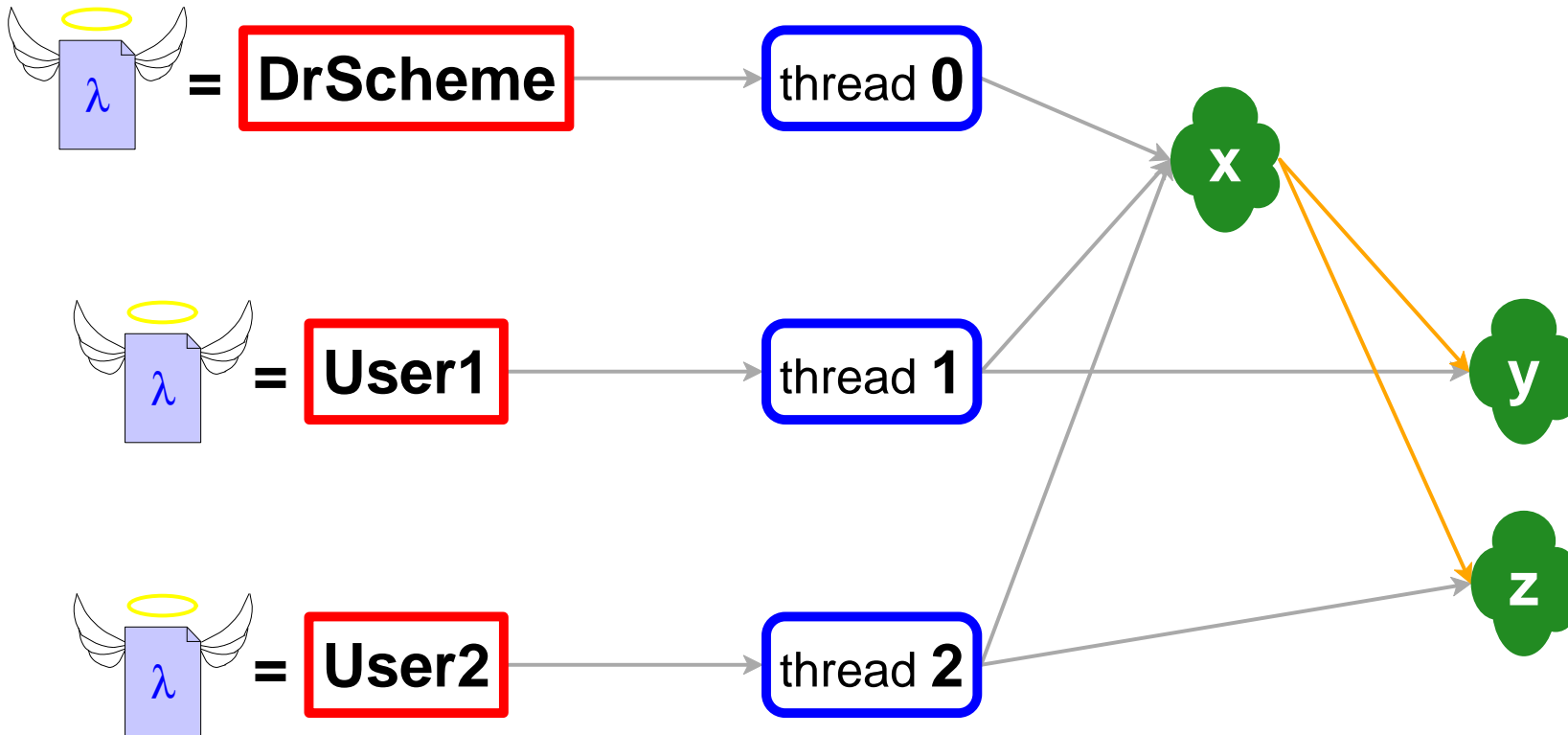
DrScheme Bug



DrScheme Repair



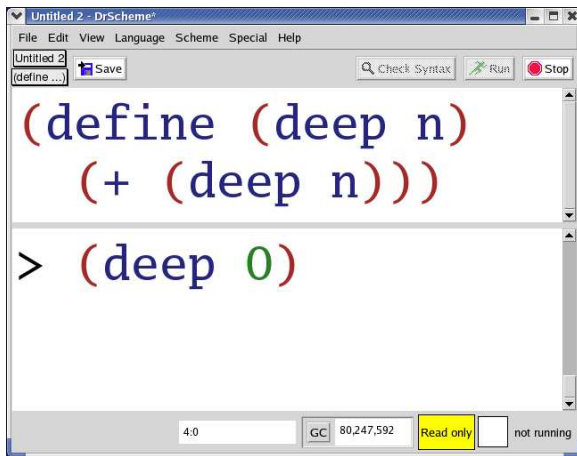
DrScheme Repair



Changed 5 references:

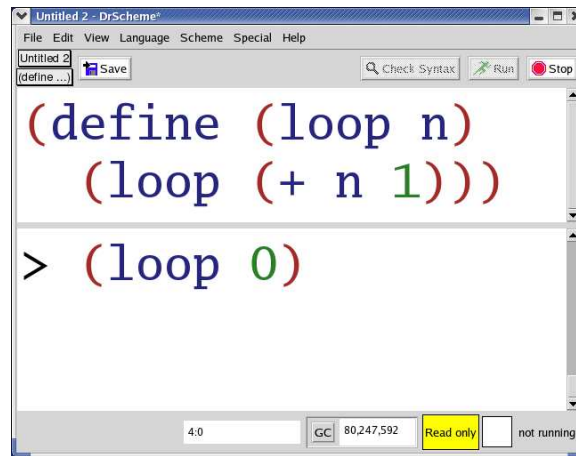
- Weakened 2
- Removed 2
- Moved 1 into child

Current Experience: DrScheme



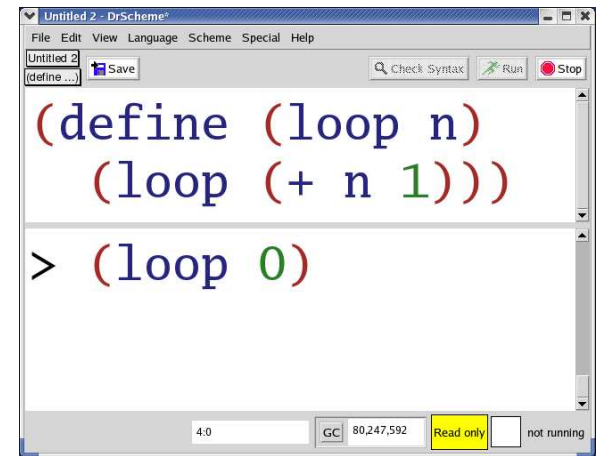
```
File Edit View Language Scheme Special Help
Untitled 2 (define ...) Save Check Syntax Run Stop
(define (deep n)
  (+ (deep n)))
> (deep 0)
4.0 GC 80,247,592 Read only not running
```

Bad Loop



```
File Edit View Language Scheme Special Help
Untitled 2 (define ...) Save Check Syntax Run Stop
(define (loop n)
  (loop (+ n 1)))
> (loop 0)
4.0 GC 80,247,592 Read only not running
```

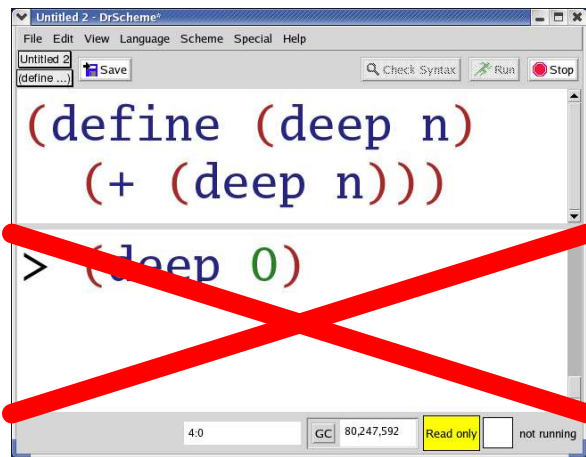
Normal



```
File Edit View Language Scheme Special Help
Untitled 2 (define ...) Save Check Syntax Run Stop
(define (loop n)
  (loop (+ n 1)))
> (loop 0)
4.0 GC 80,247,592 Read only not running
```

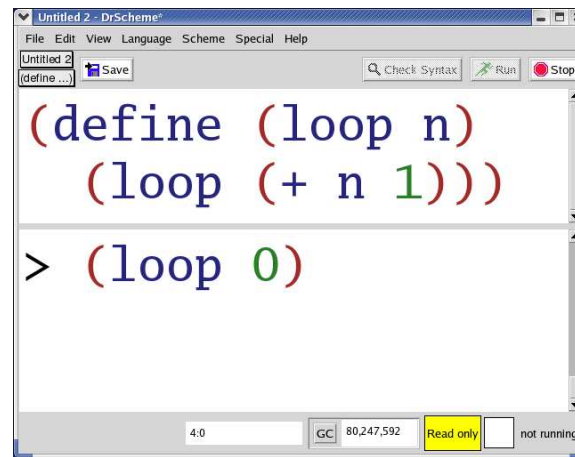
Normal

Current Experience: DrScheme



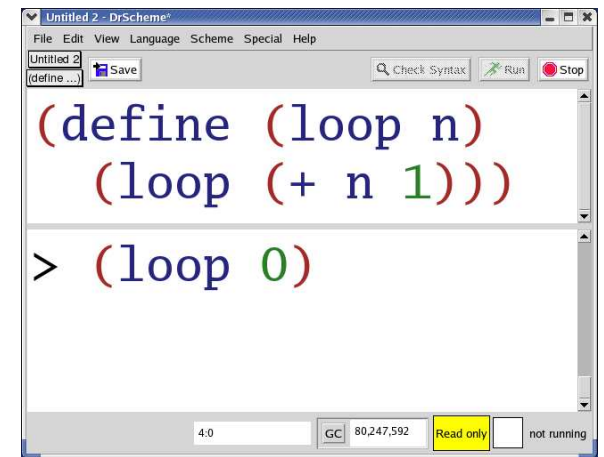
```
(define (deep n)
  (+ (deep n)))
> (deep 0)
```

Shut Down



```
(define (loop n)
  (loop (+ n 1)))
> (loop 0)
```

Normal



```
(define (loop n)
  (loop (+ n 1)))
> (loop 0)
```

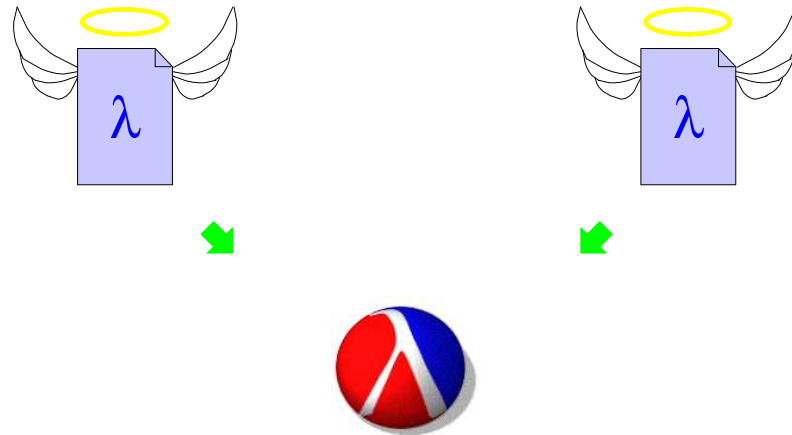
Normal

Accounting without Partitions

Useful accounting

- Doesn't need partitions
- Does need hierarchy

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