

# Kill-Safe Synchronization Abstractions



**Matthew Flatt**

University of Utah

**Robert Bruce Findler**

University of Chicago

# Sibling Food-Sharing Protocol



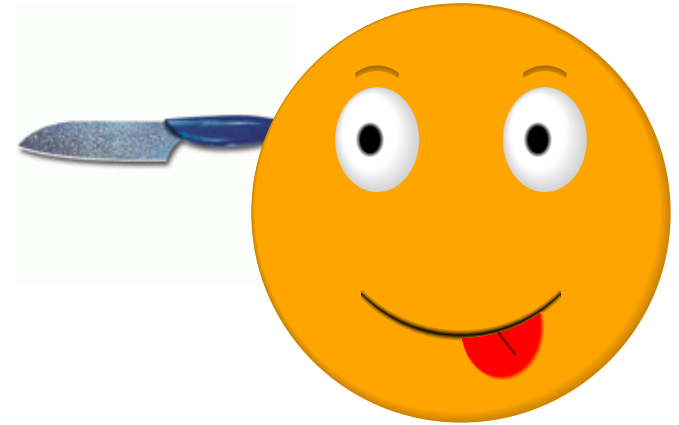
# Sibling Food-Sharing Protocol



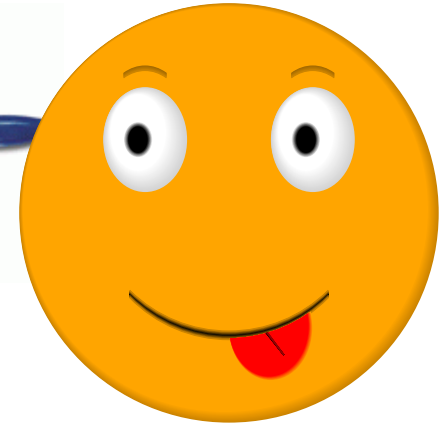
# Sibling Food-Sharing Protocol



# Sibling Food-Sharing Protocol



# Sibling Food-Sharing Protocol



# Sibling Food-Sharing Protocol



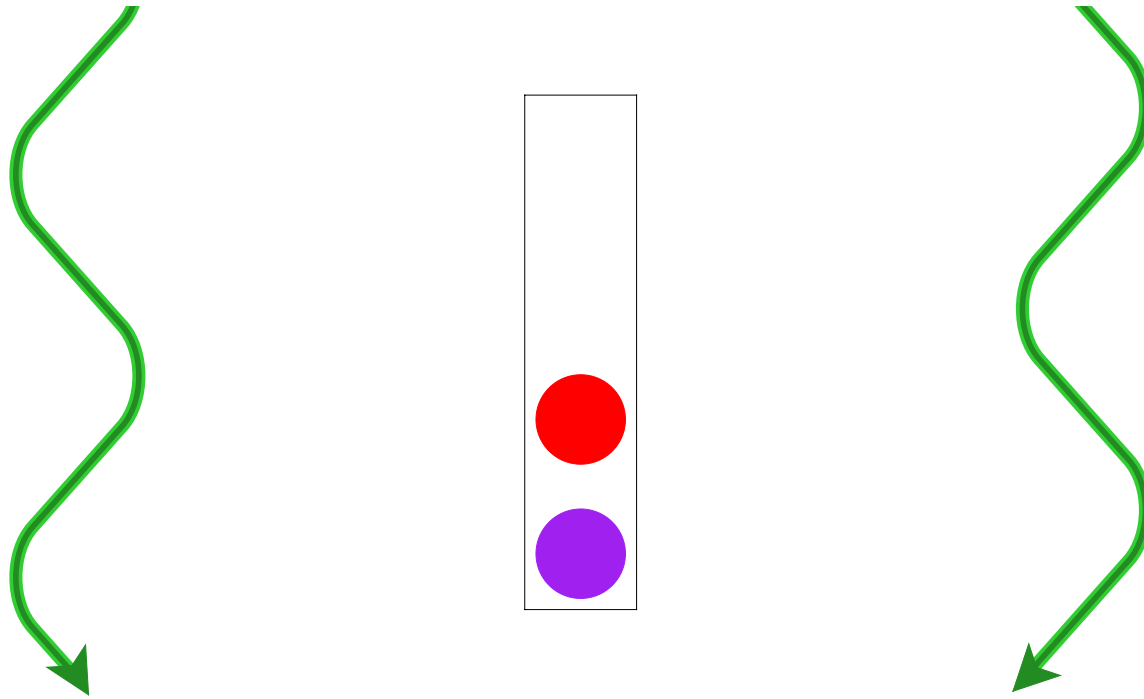
# Sibling Food-Sharing Protocol



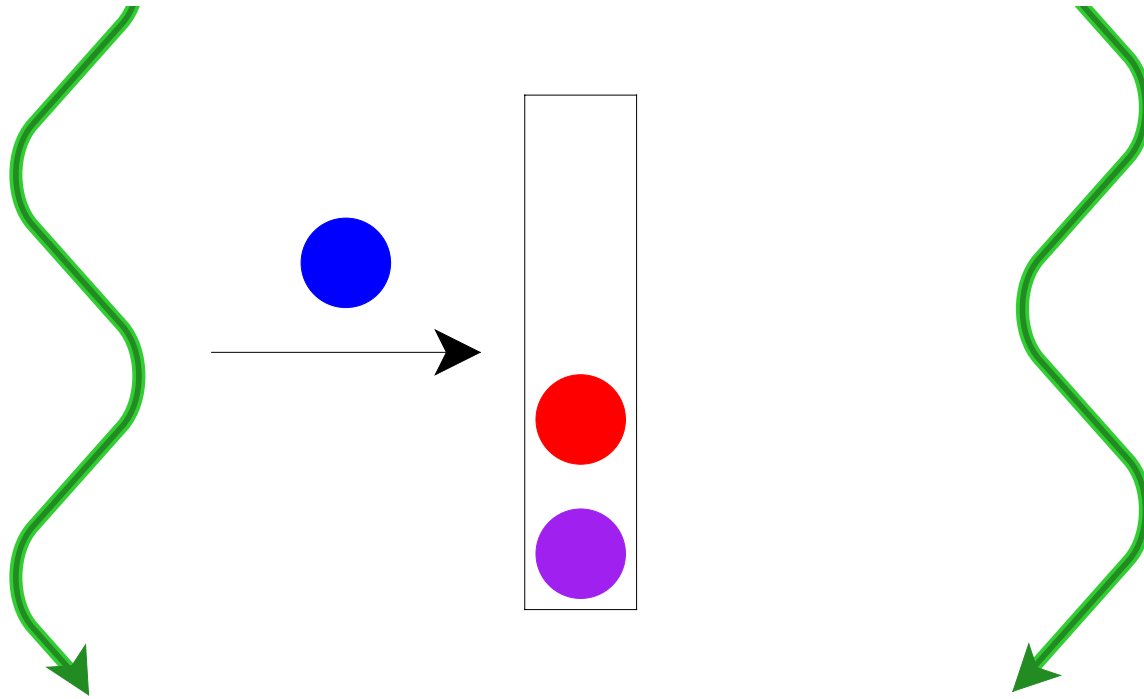
- By inspection, the protocol is fair
- No parental supervision required



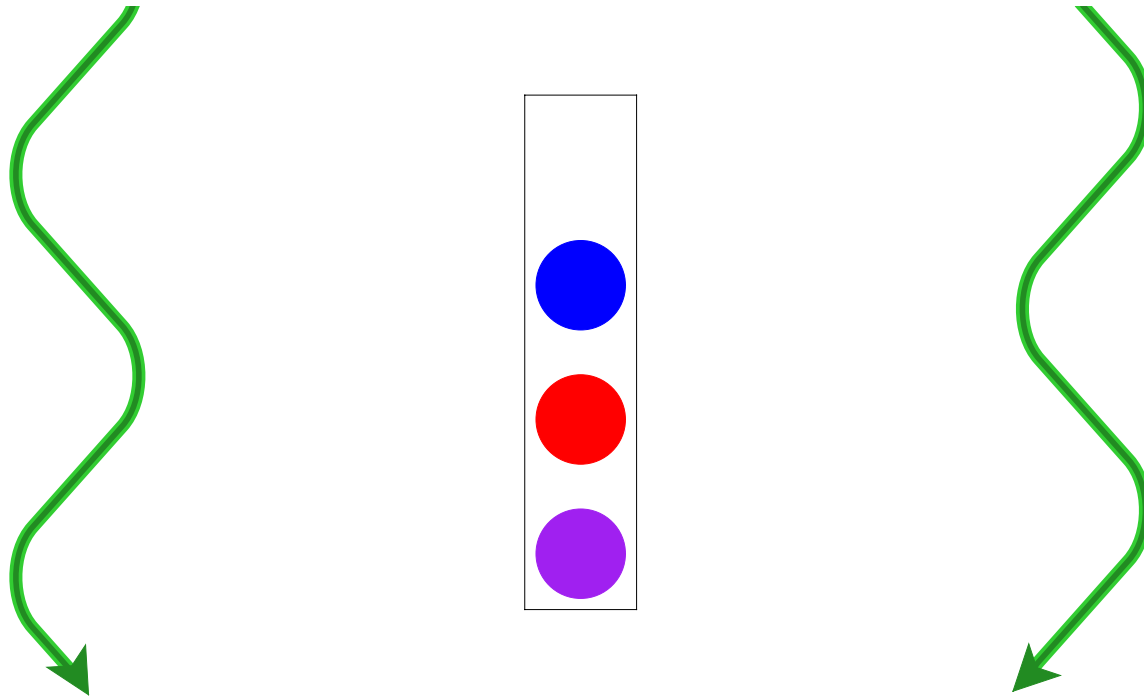
# Sharing among Processes



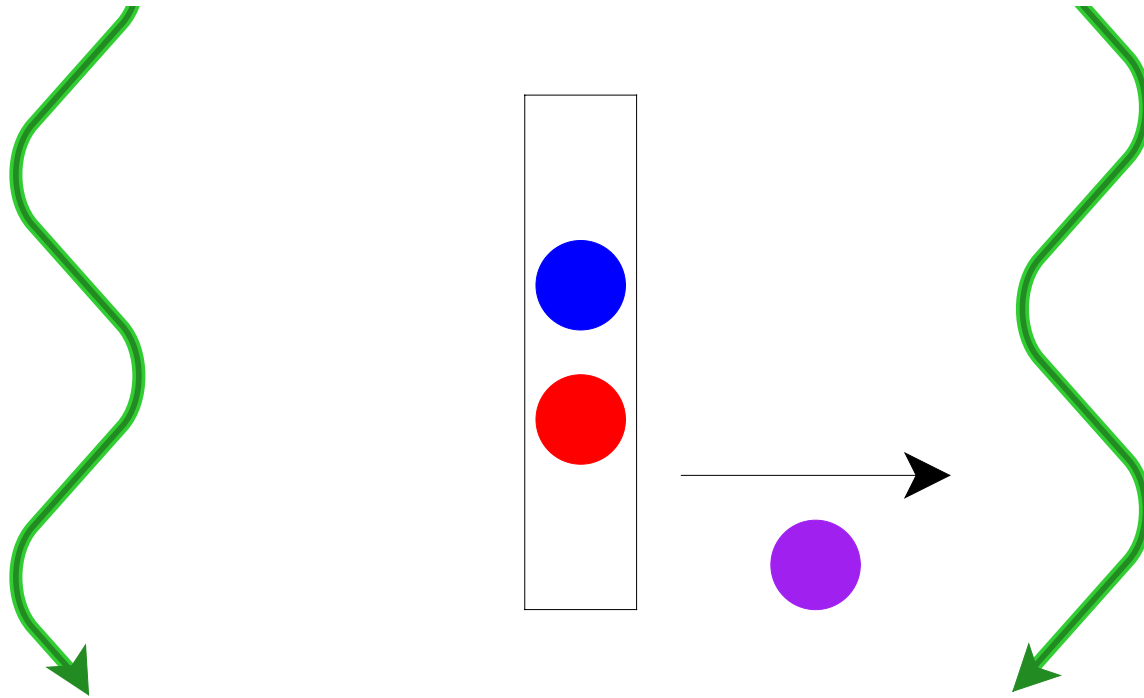
# Sharing among Processes



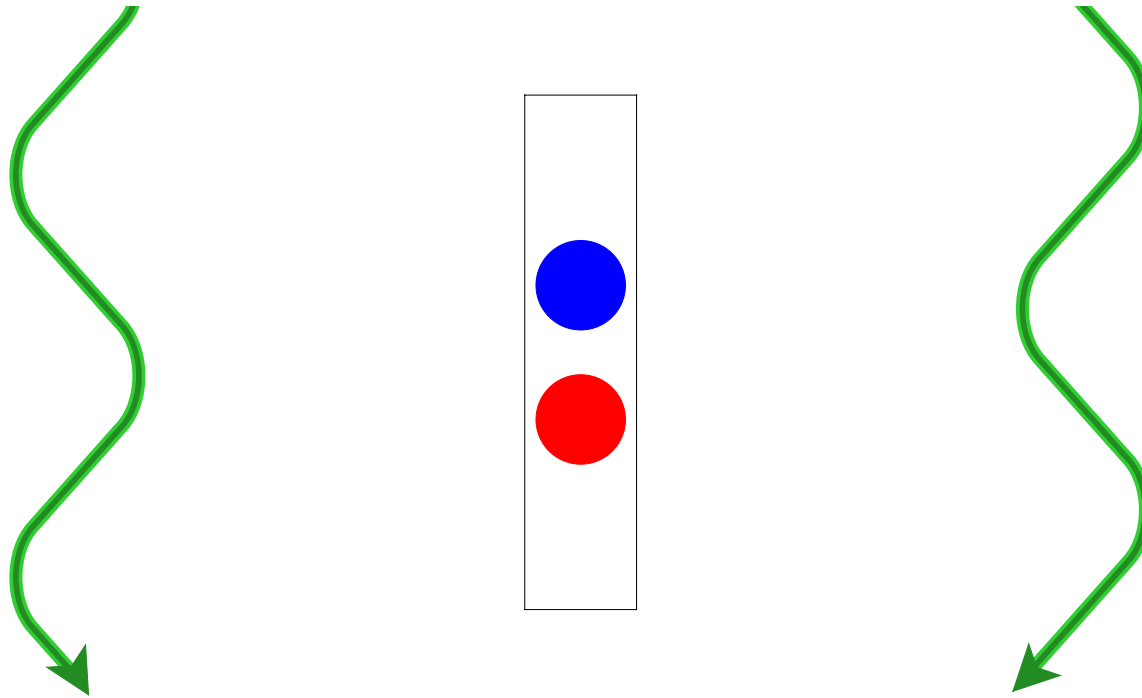
# Sharing among Processes



# Sharing among Processes

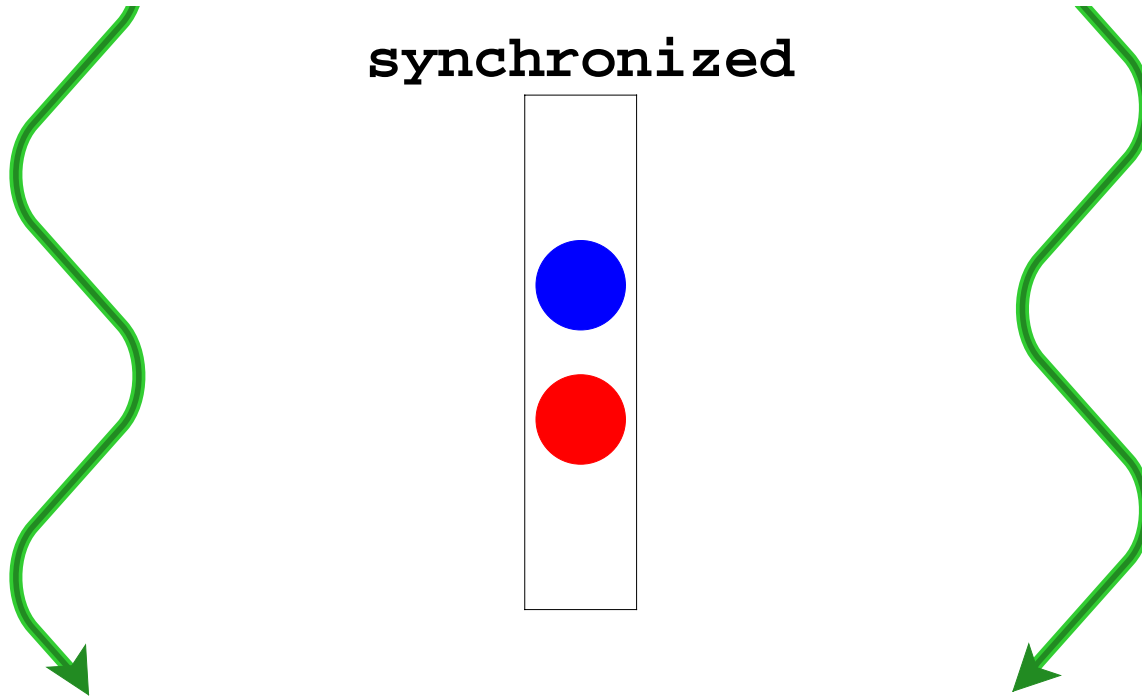


# Sharing among Processes

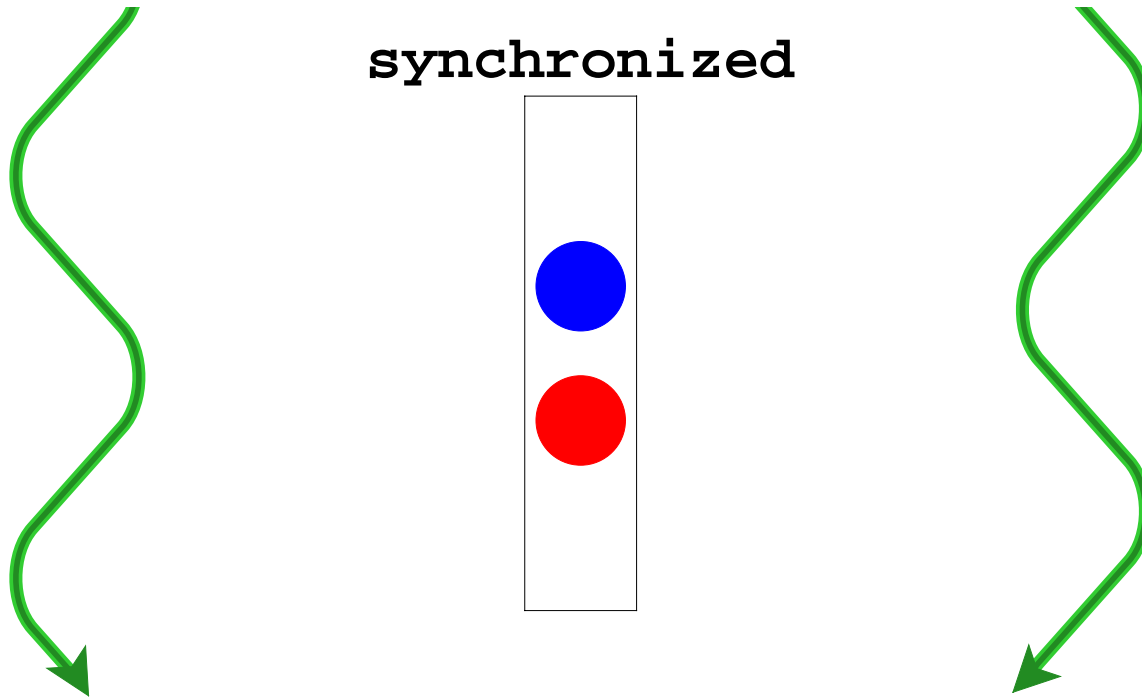


- Queue should be safe and fair
- Should require no kernel supervision

# Sharing in Java

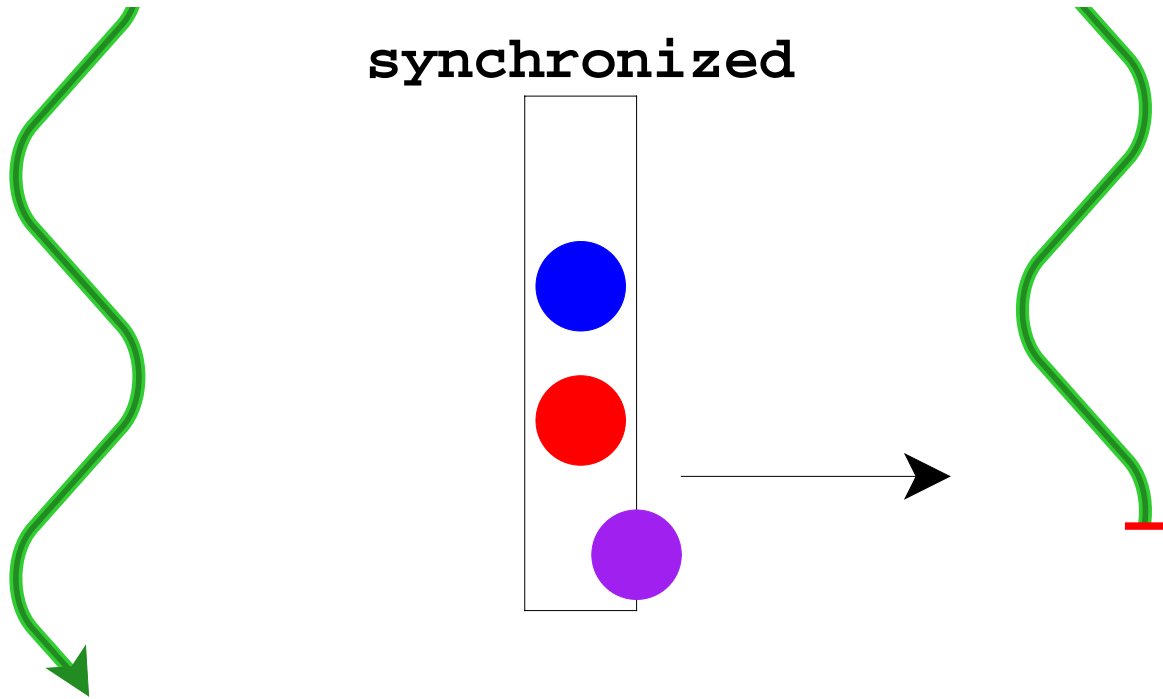


# Sharing in Java



**Thread.stop**  $\Rightarrow$  **synchronized** *isn't enough*

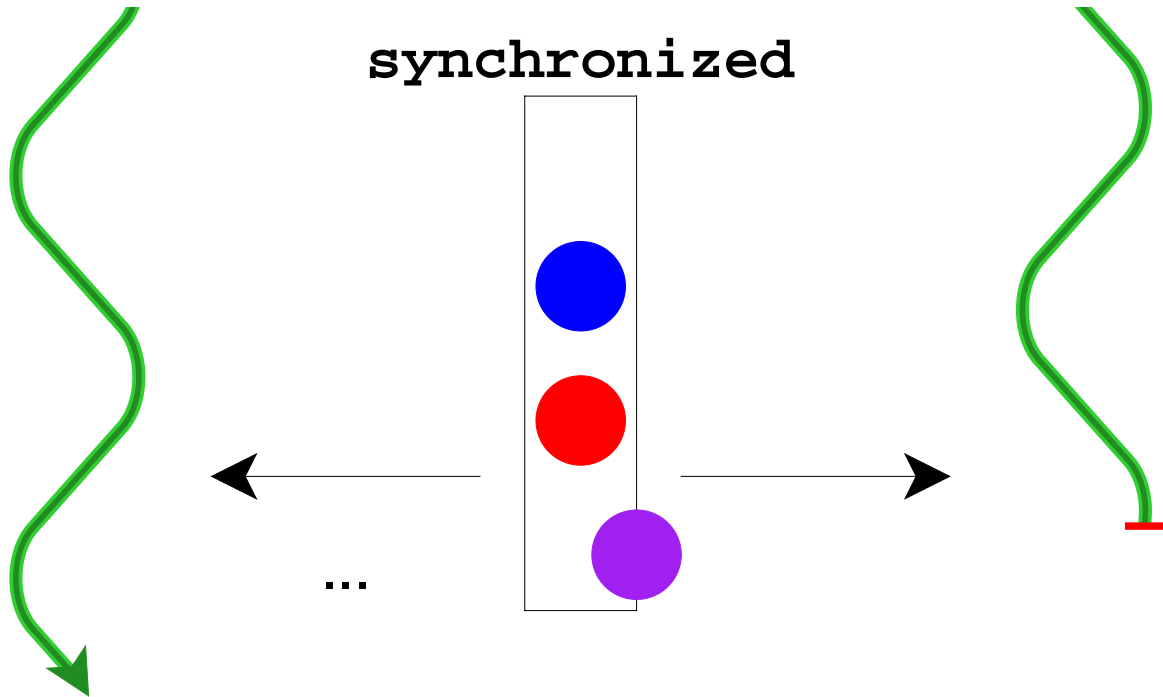
# Sharing in Java



**Thread.stop**  $\Rightarrow$  **synchronized** *isn't enough*

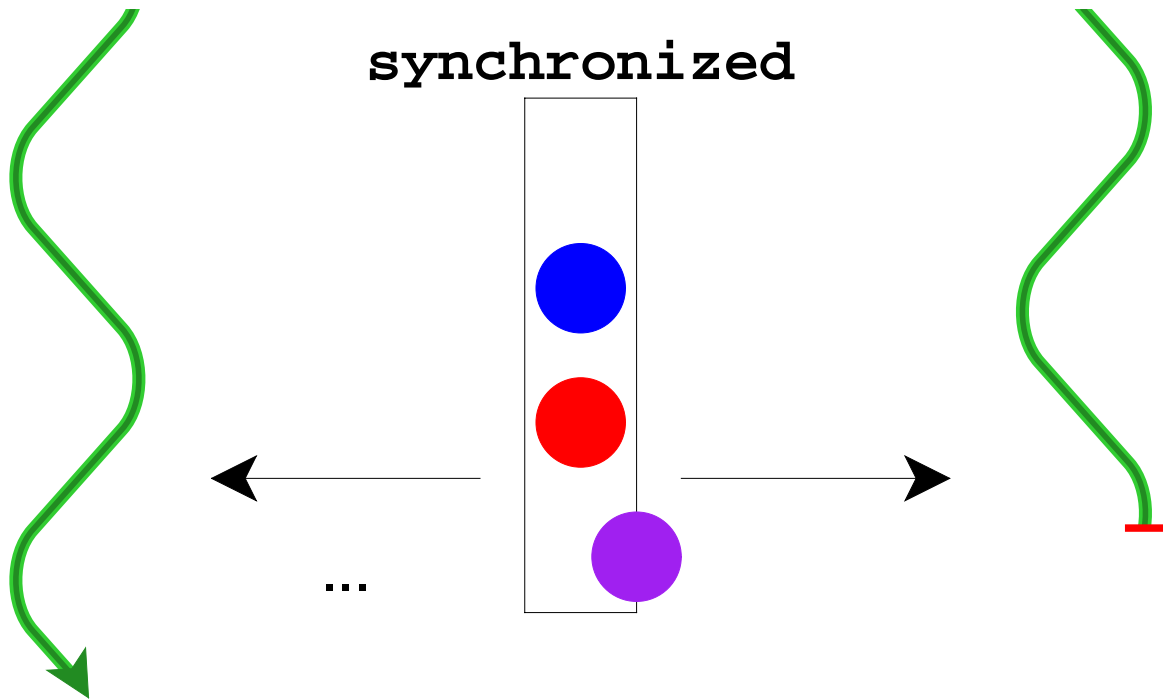


# Sharing in Java



**Thread.stop**  $\Rightarrow$  **synchronized** *isn't enough*

# Sharing in Java

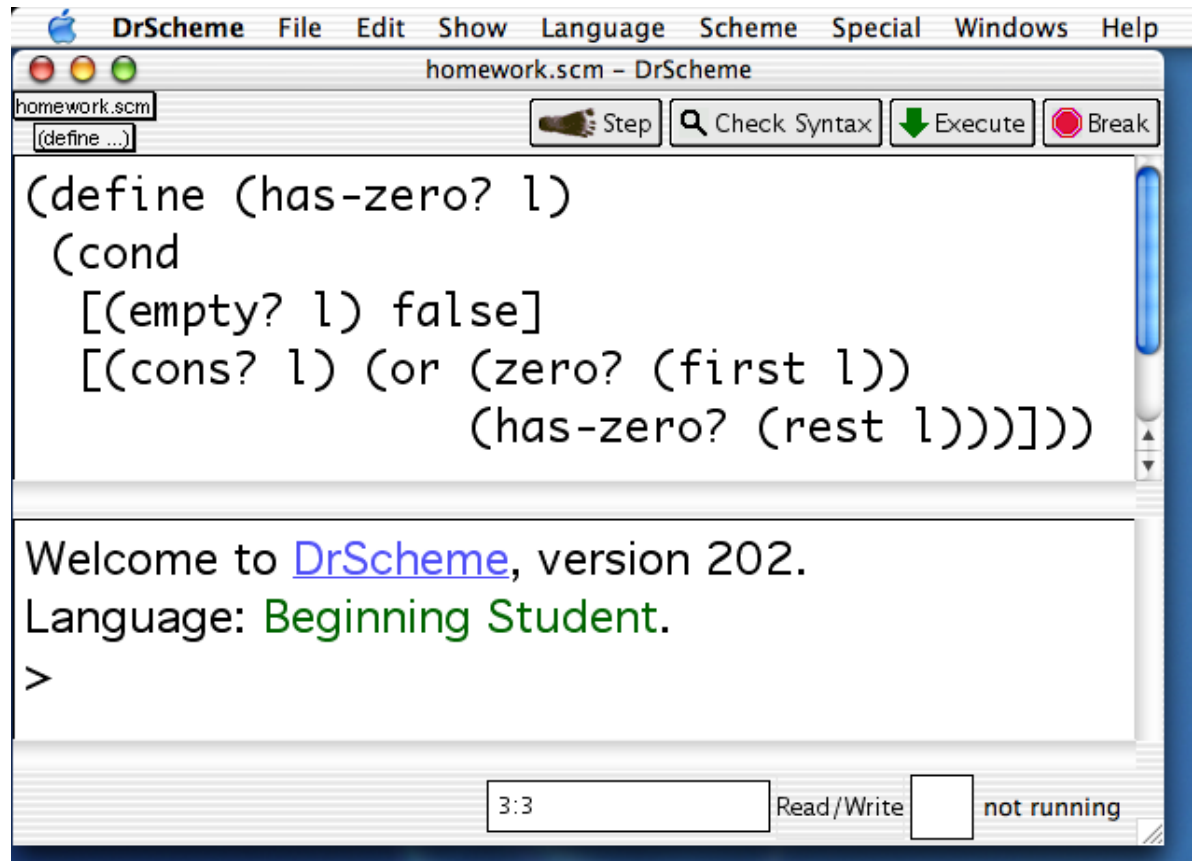


**Thread.stop**  $\Rightarrow$  **synchronized** *isn't enough*

$\therefore$  Java has no **Thread.stop**

# Why Terminate?

- Execute code in a programming environment (DrScheme)



The screenshot shows the DrScheme interface. The menu bar includes DrScheme, File, Edit, Show, Language, Scheme, Special, Windows, and Help. The window title is "homework.scm - DrScheme". The editor contains the following Scheme code:

```
(define (has-zero? l)
  (cond
    [(empty? l) false]
    [(cons? l) (or (zero? (first l))
                   (has-zero? (rest l)))]))
```

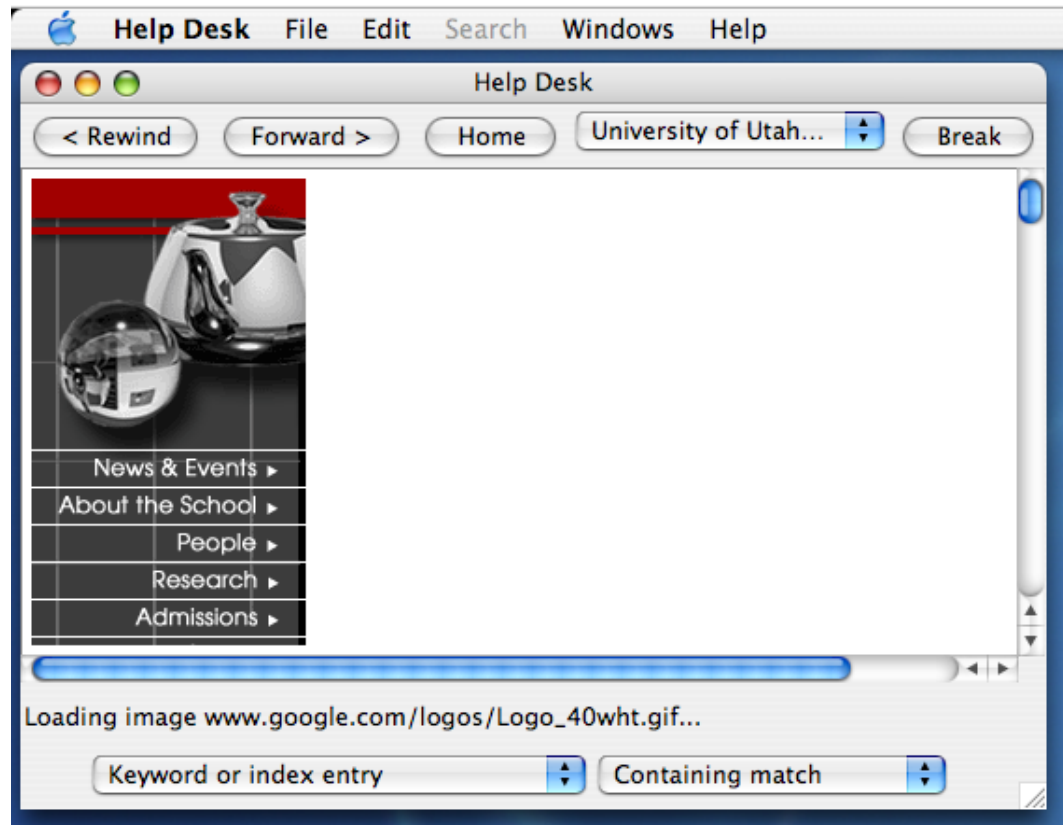
Below the editor, the console displays a welcome message:

```
Welcome to DrScheme, version 202.
Language: Beginning Student.
>
```

At the bottom, the status bar shows "3:3" in a text box, "Read/Write" with a checkbox, and "not running" with a checkbox.

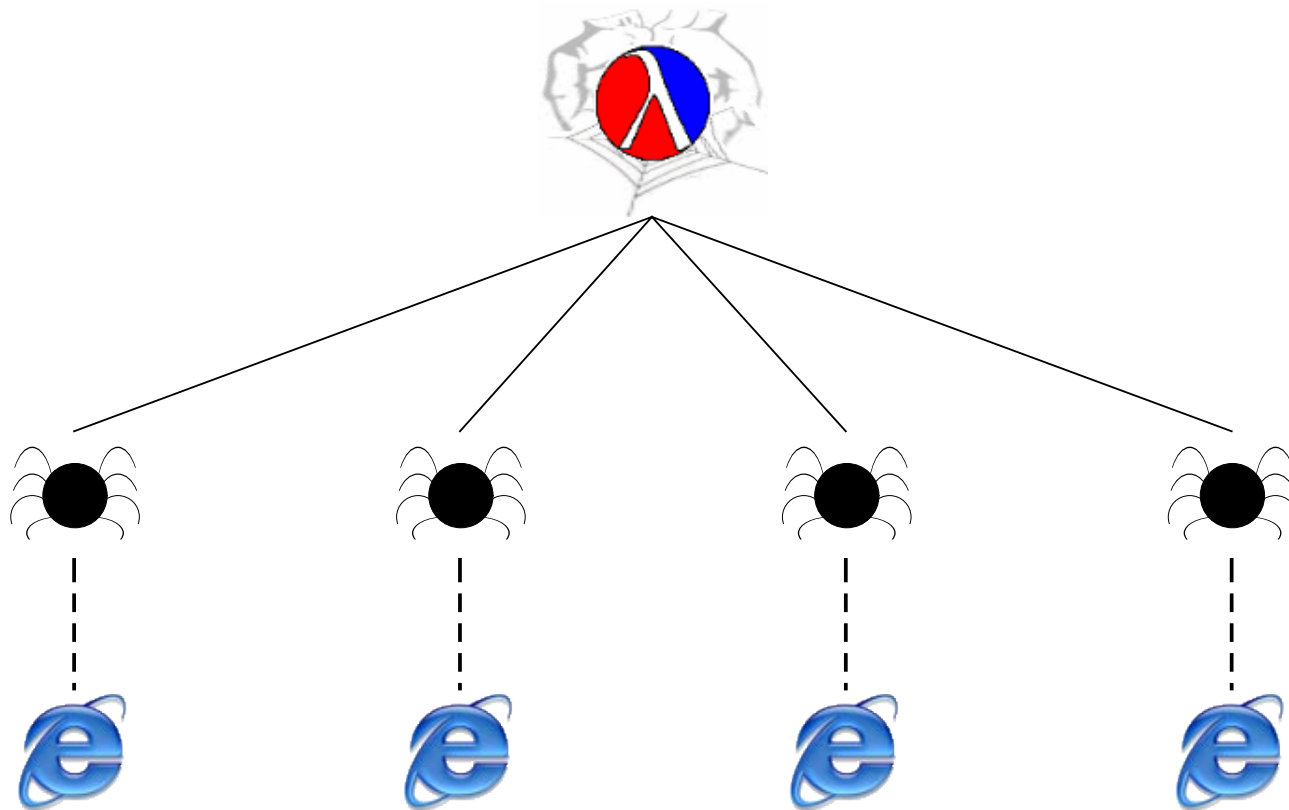
# Why Terminate?

- Execute code in a programming environment (DrScheme)
- Cancel actions that allocate resources (HTML browser)

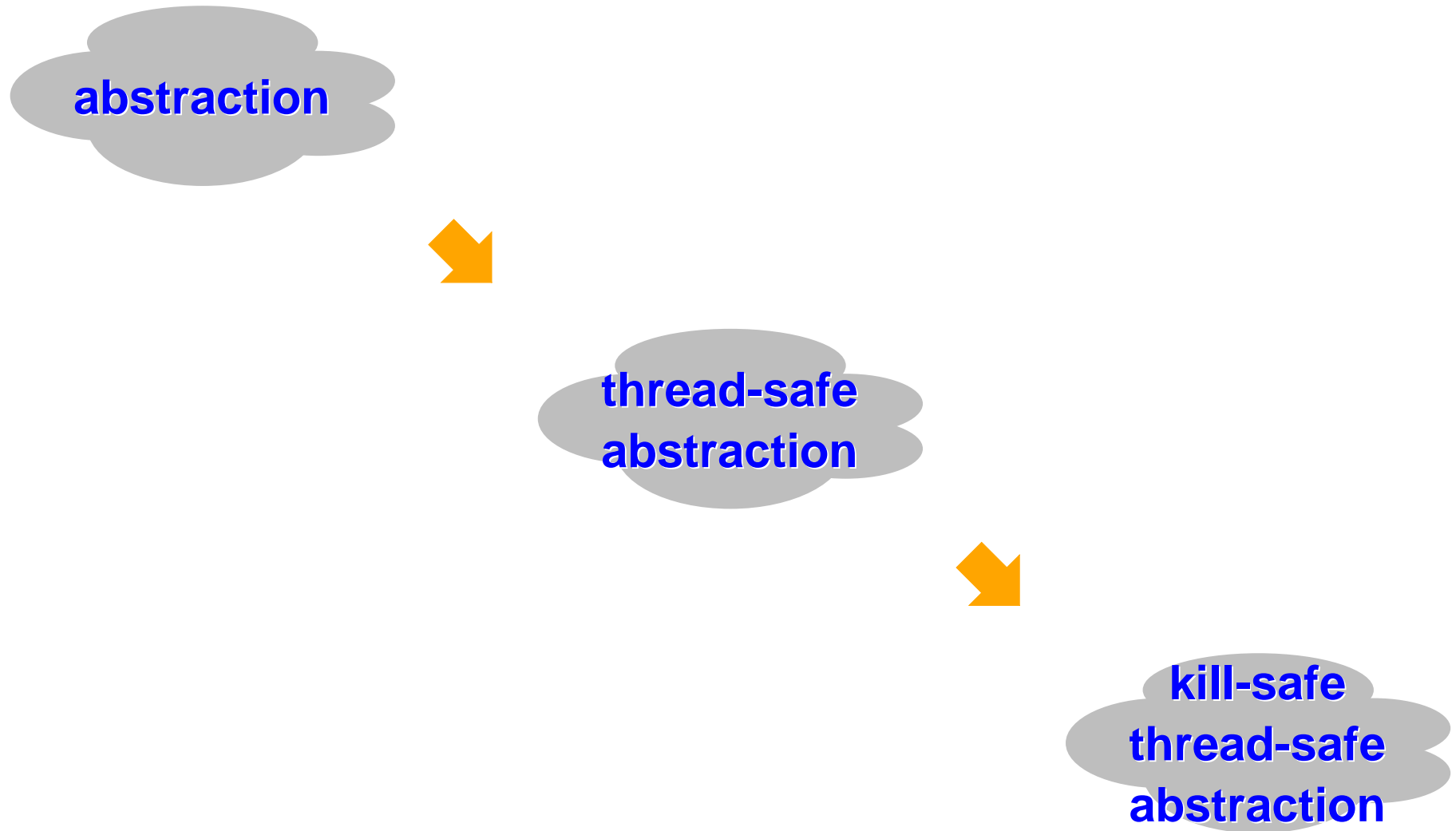


# Why Terminate?

- Execute code in a programming environment (DrScheme)
- Cancel actions that allocate resources (HTML browser)
- Stop misbehaving servlets (web server)



# Building Kill-Safe Abstractions



# Building Kill-Safe Abstractions

**abstraction**

Programmer effort  
– but generally understood

**thread-safe  
abstraction**

**kill-safe  
thread-safe  
abstraction**

# Building Kill-Safe Abstractions

**abstraction**

Programmer effort  
– but generally understood

**thread-safe  
abstraction**

Programmer effort  
– the subject of this talk

**kill-safe  
thread-safe  
abstraction**



# Building Kill-Safe Abstractions

abstraction

Start with **Concurrent ML**  
[Reppy 88]

thread-safe  
abstraction

kill-safe  
thread-safe  
abstraction

# Building Kill-Safe Abstractions

abstraction

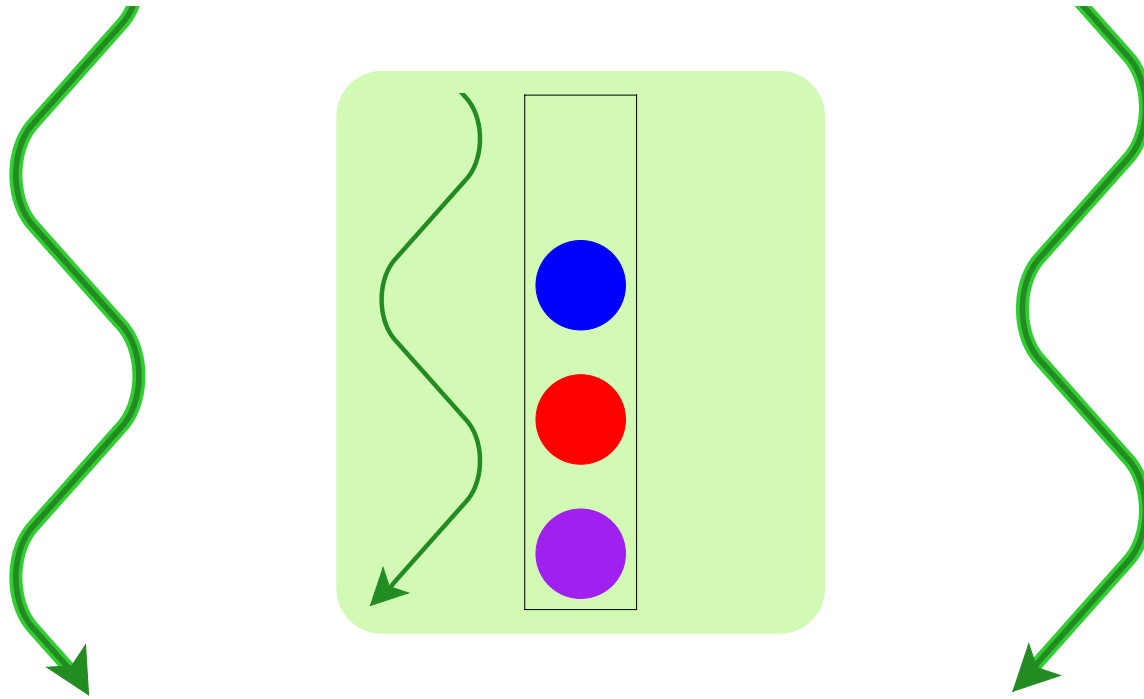
Start with **Concurrent ML**  
[Reppy 88]

thread-safe  
abstraction

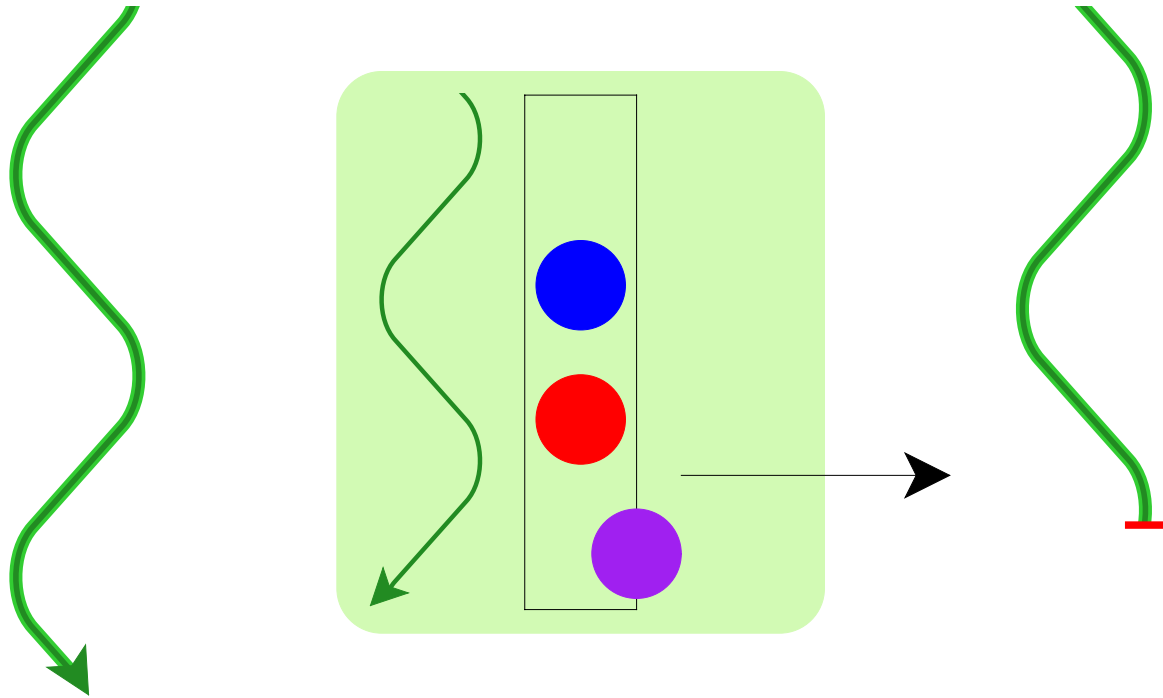
Add MzScheme's **custodians**  
and a little more

kill-safe  
thread-safe  
abstraction

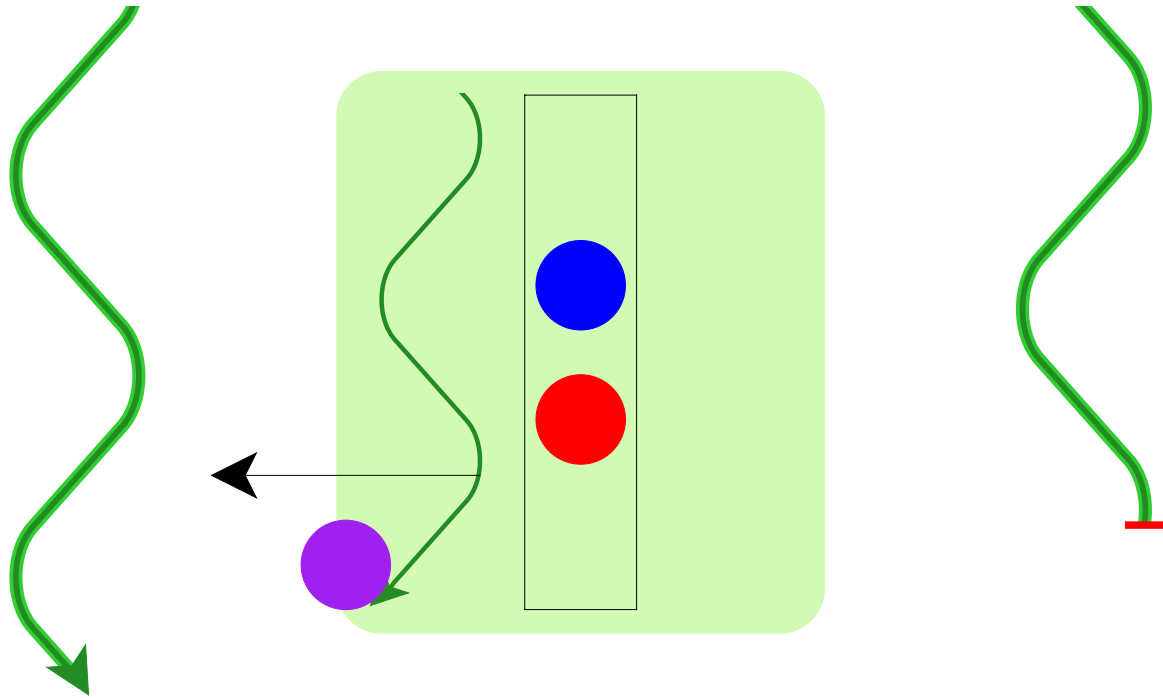
# Sharing in Concurrent ML



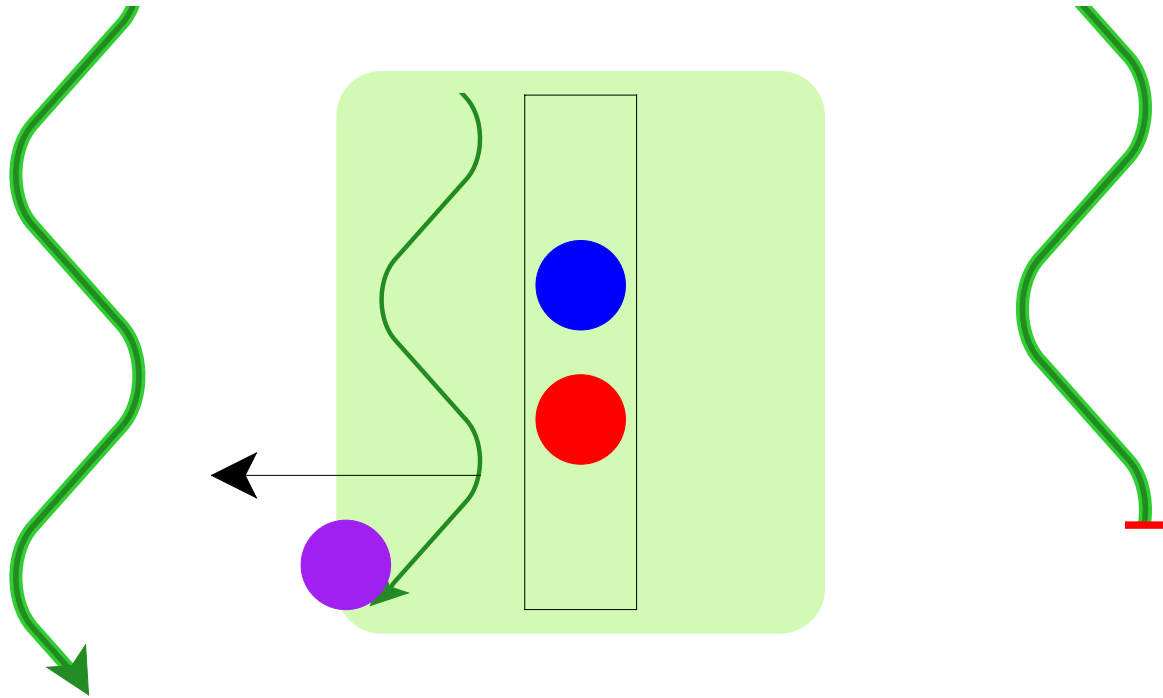
# Sharing in Concurrent ML



# Sharing in Concurrent ML

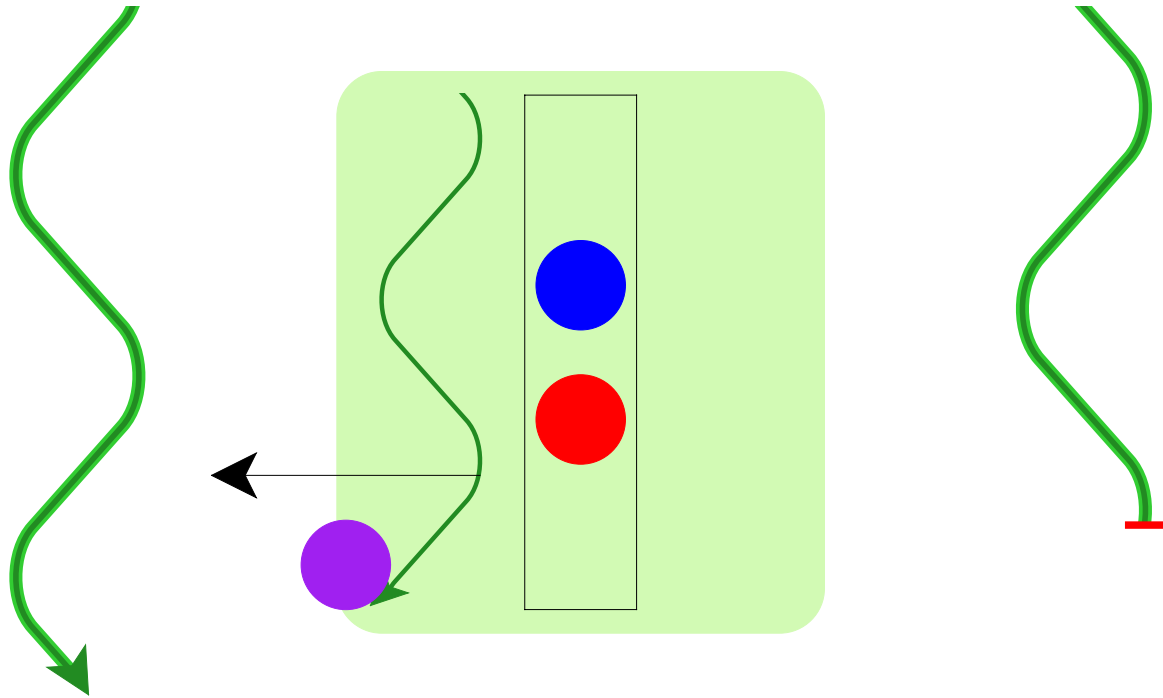


# Sharing in Concurrent ML



Abstraction-as-process naturally supports termination

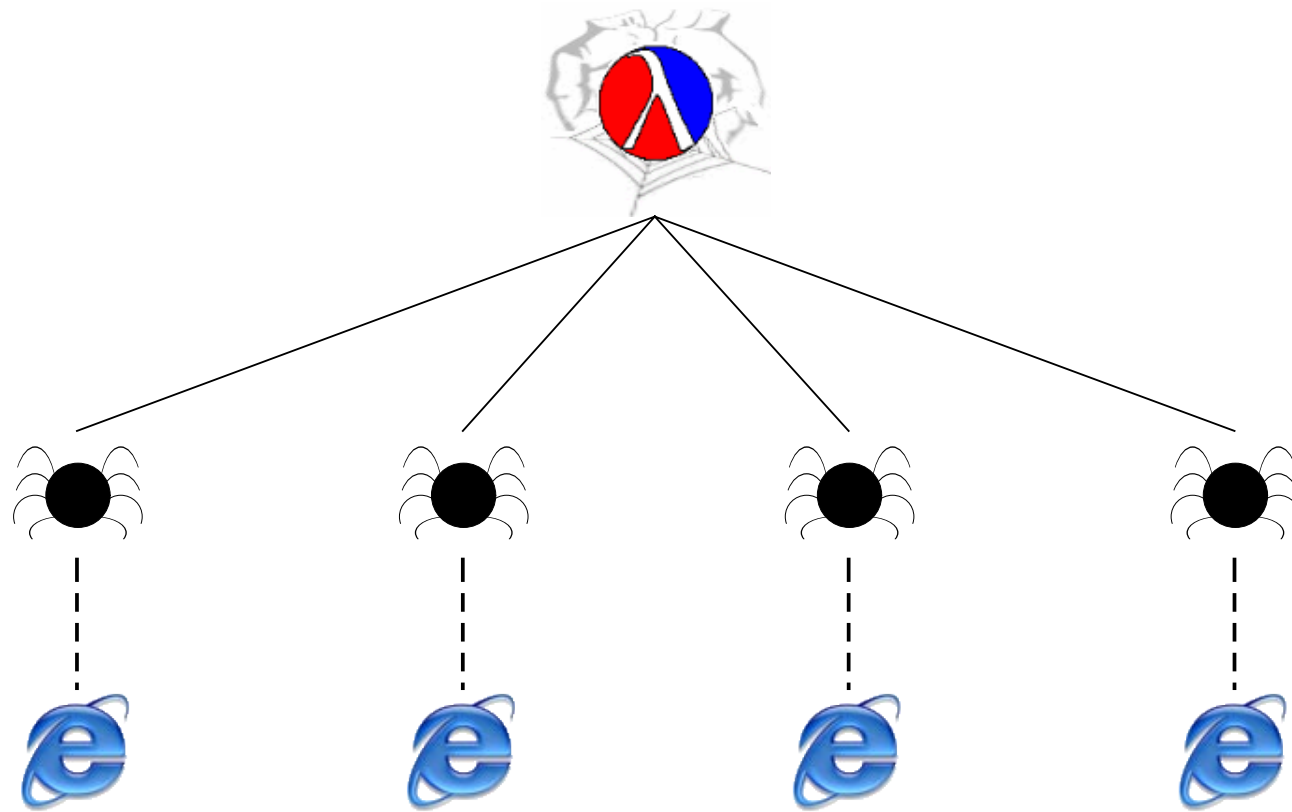
# Sharing in Concurrent ML



Abstraction-as-process naturally supports termination

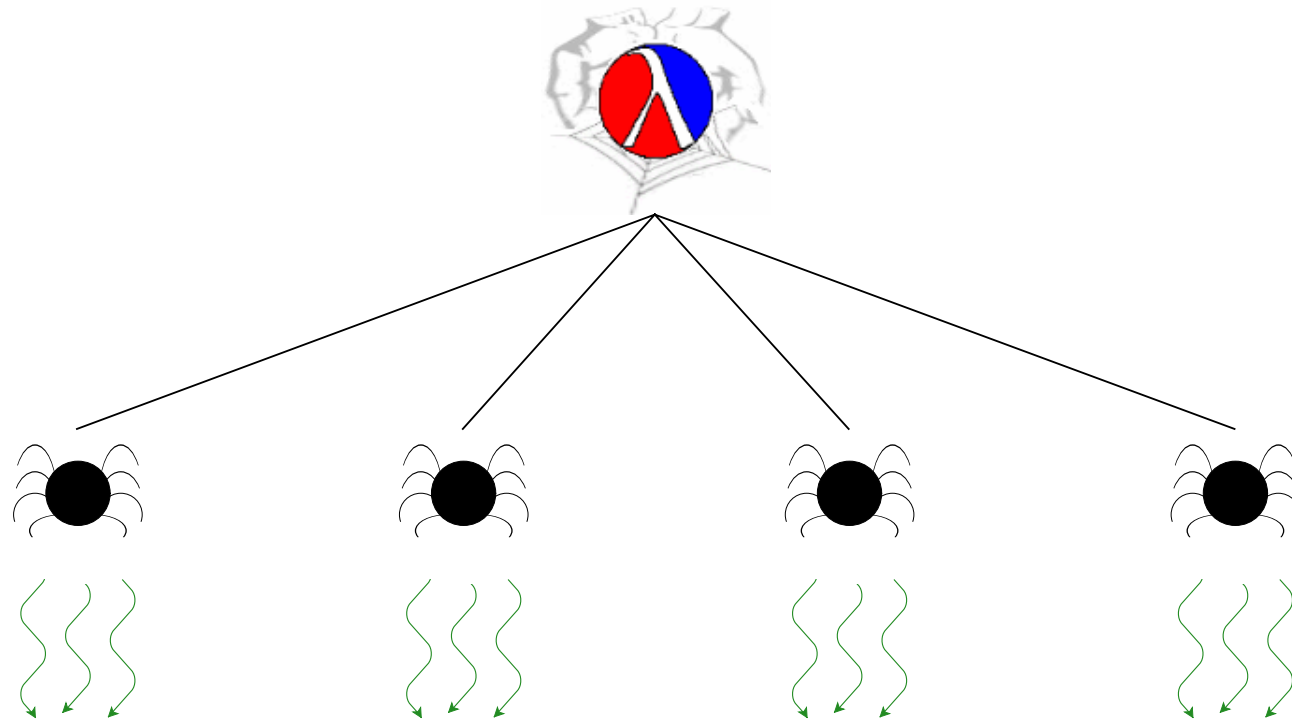
Remaining problem: who controls the abstraction's process?

# Managing Processes and Threads

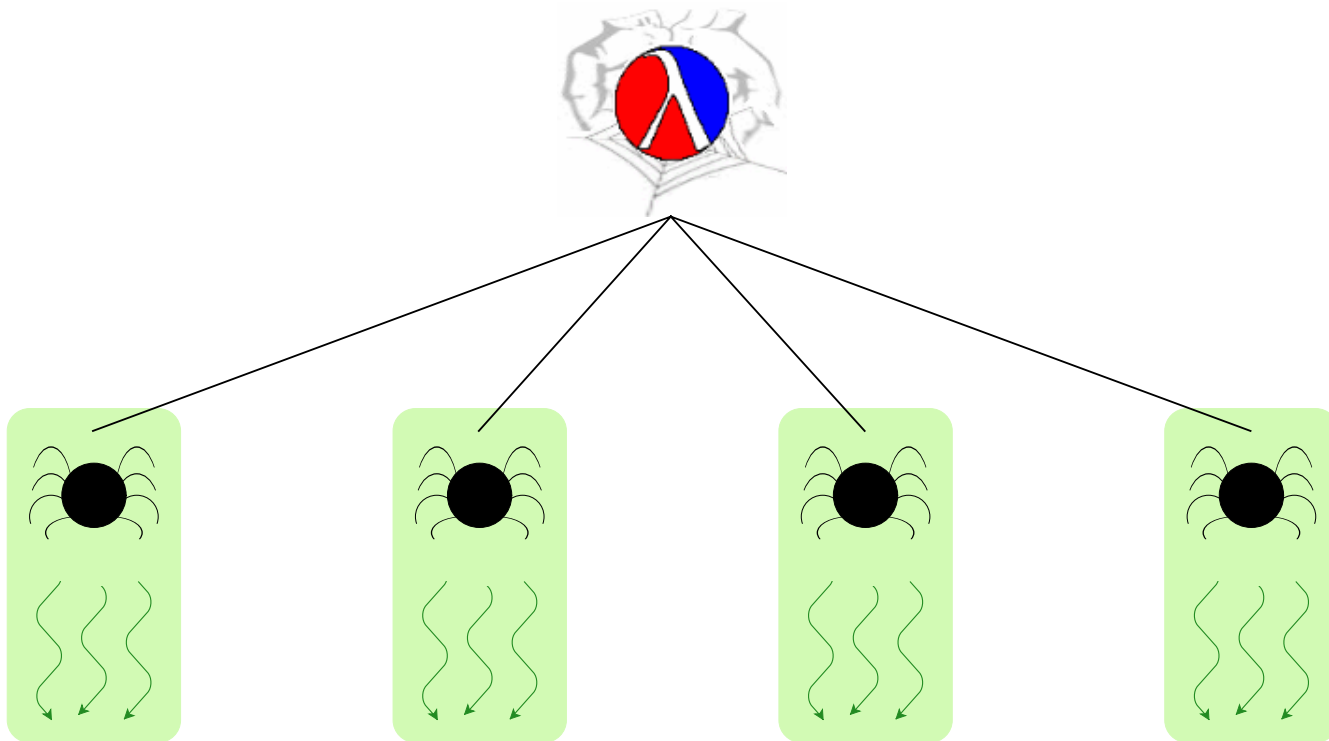




# Managing Processes and Threads

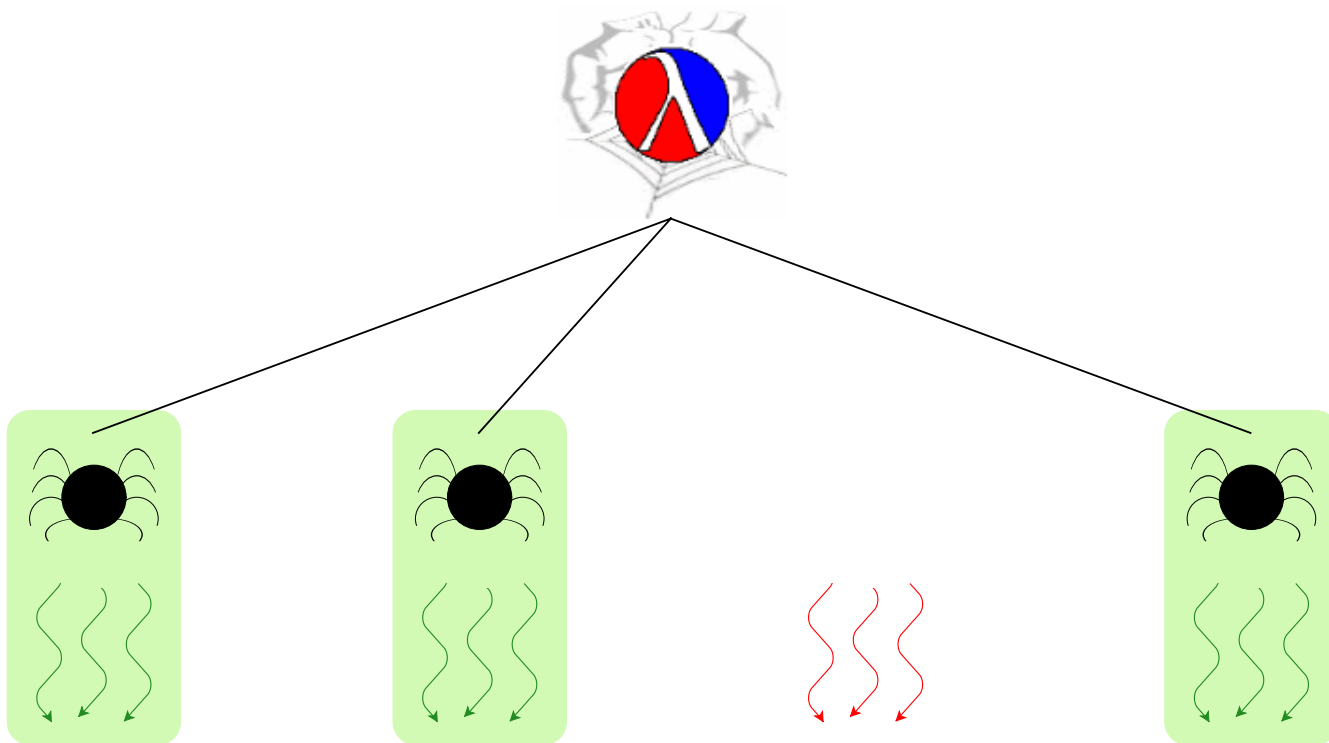


# Managing Processes and Threads



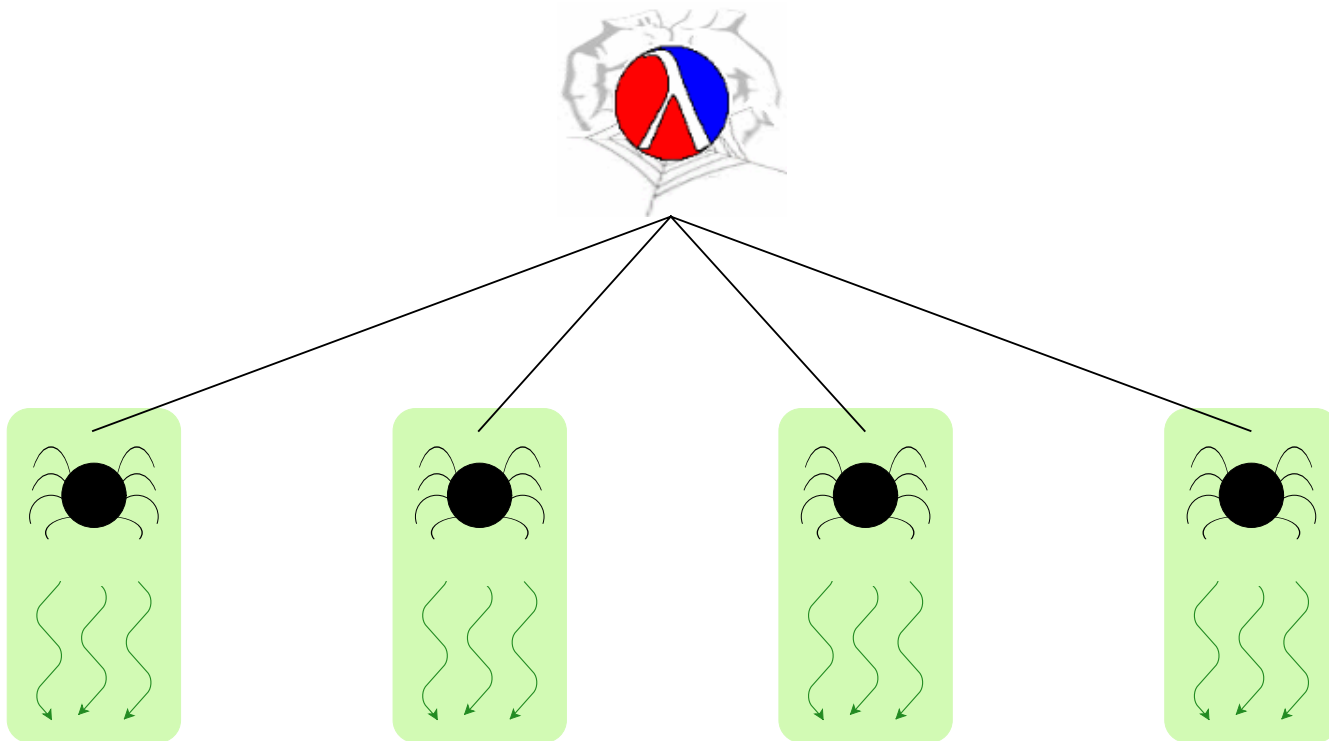
 = **custodian** = capability to execute

# Managing Processes and Threads

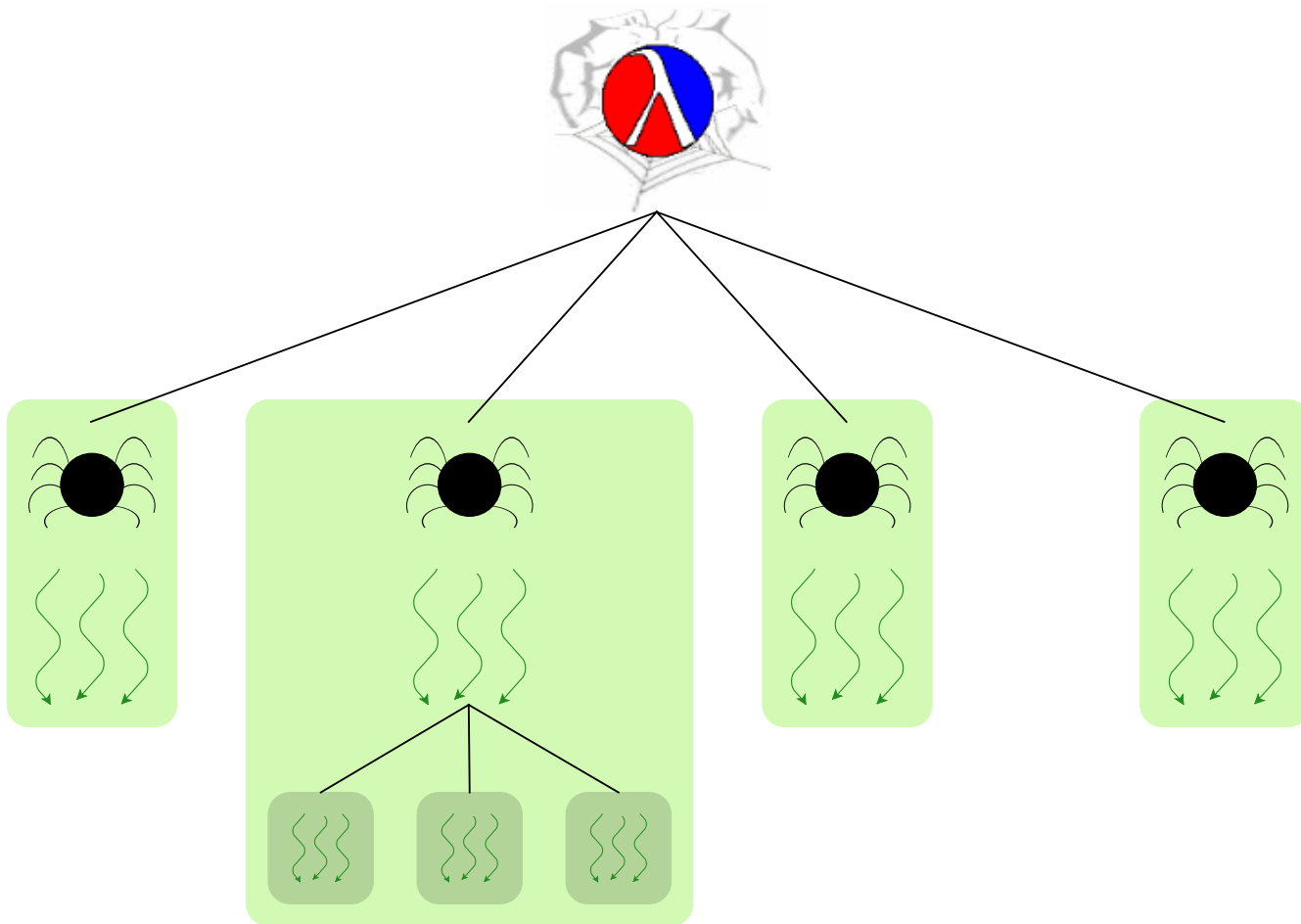


 = **custodian** = capability to execute

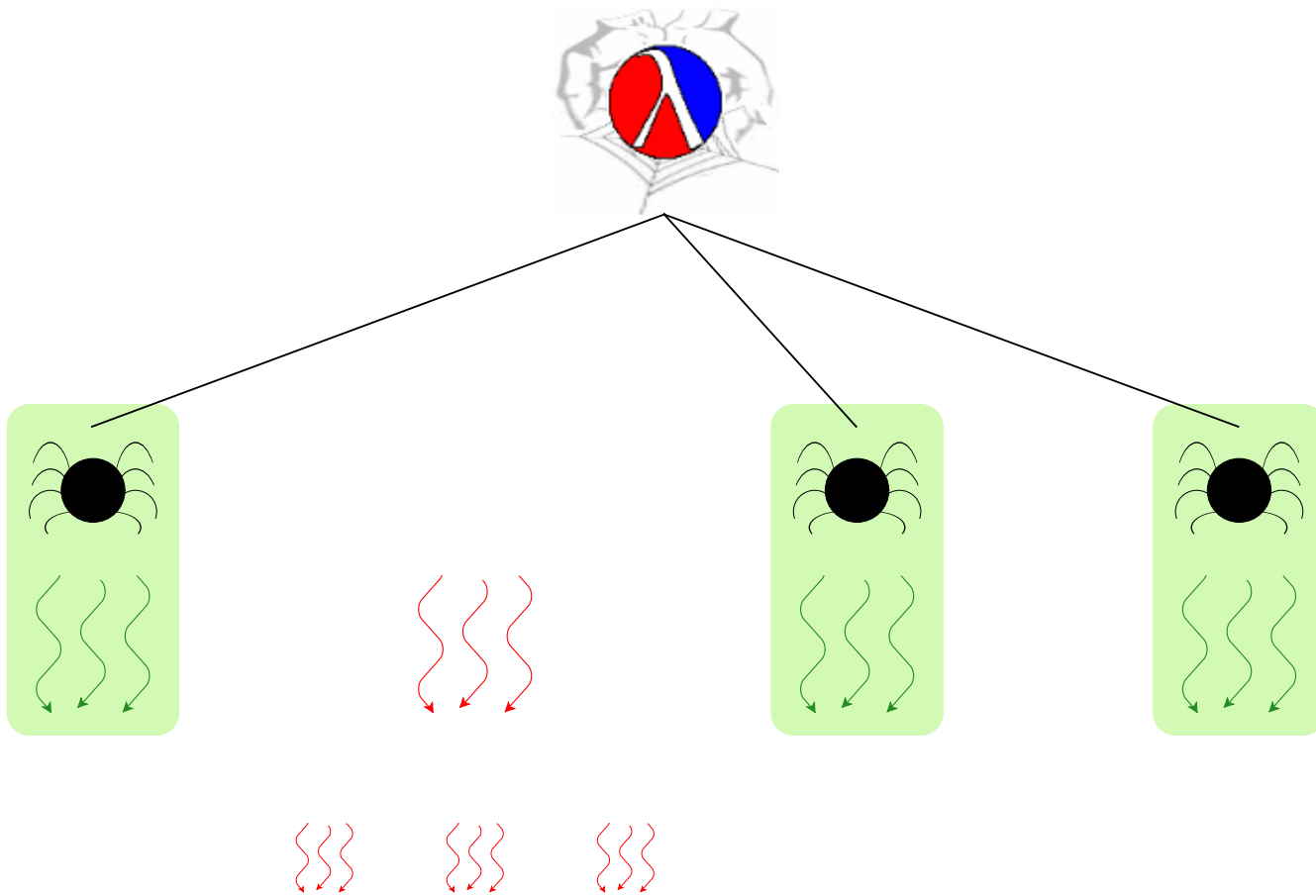
# Managing with Custodians



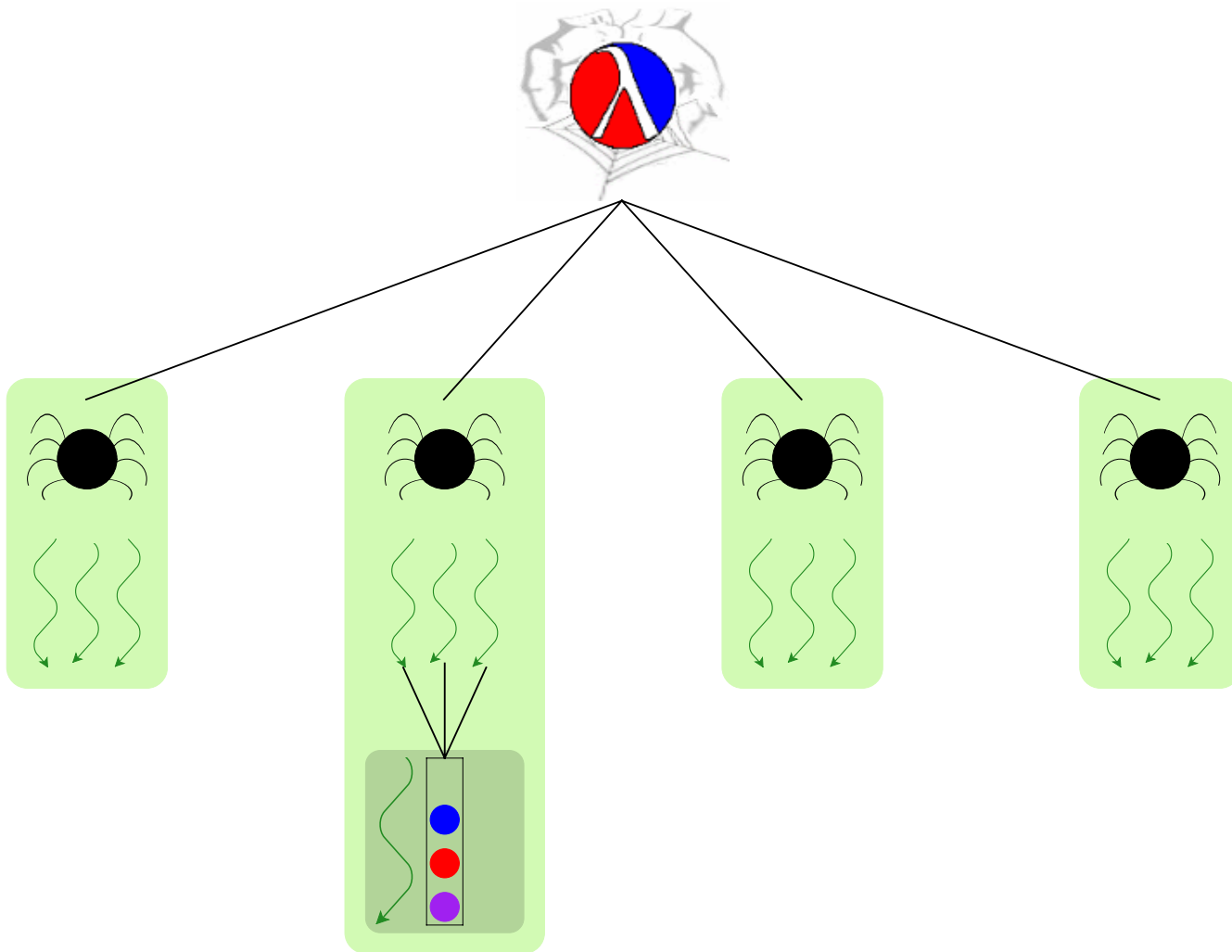
# Managing with Custodians



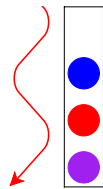
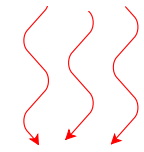
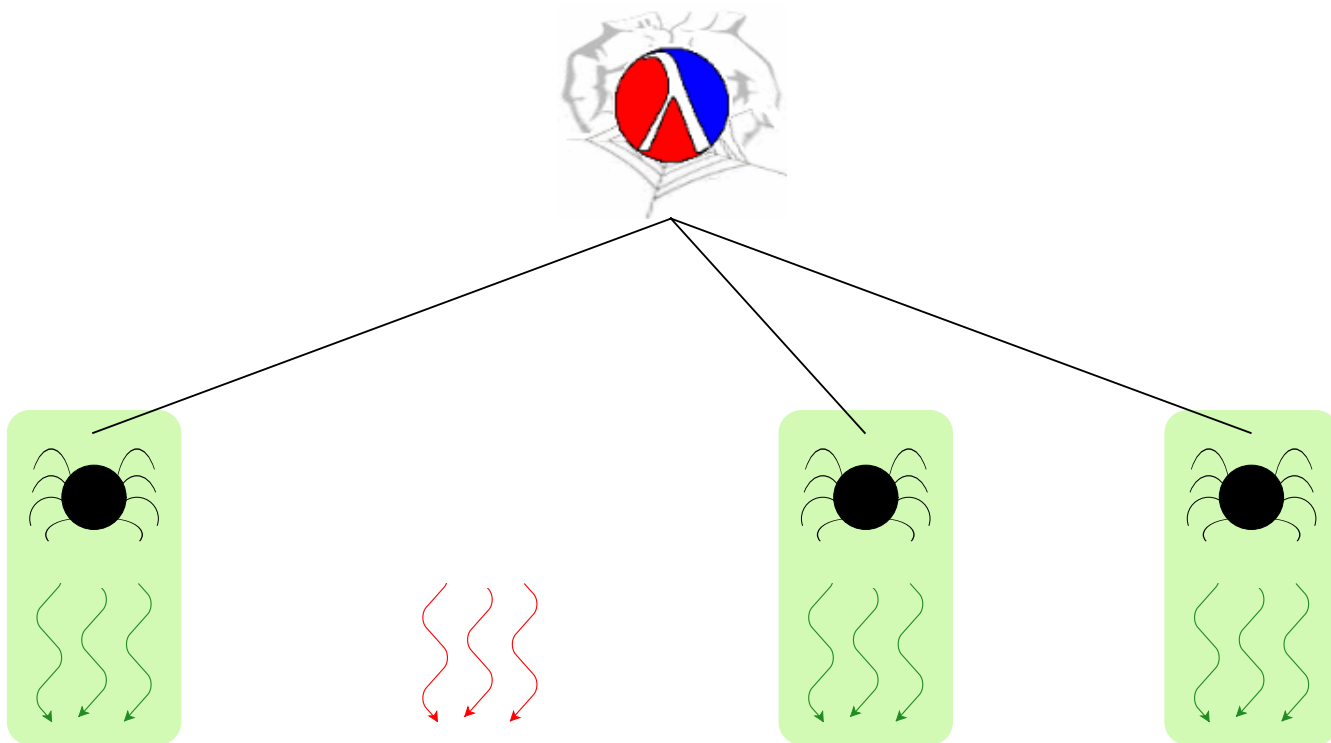
# Managing with Custodians



# Managing with Custodians



# Managing with Custodians



**Queue terminated with servlet**





# Thread-Safe Abstractions

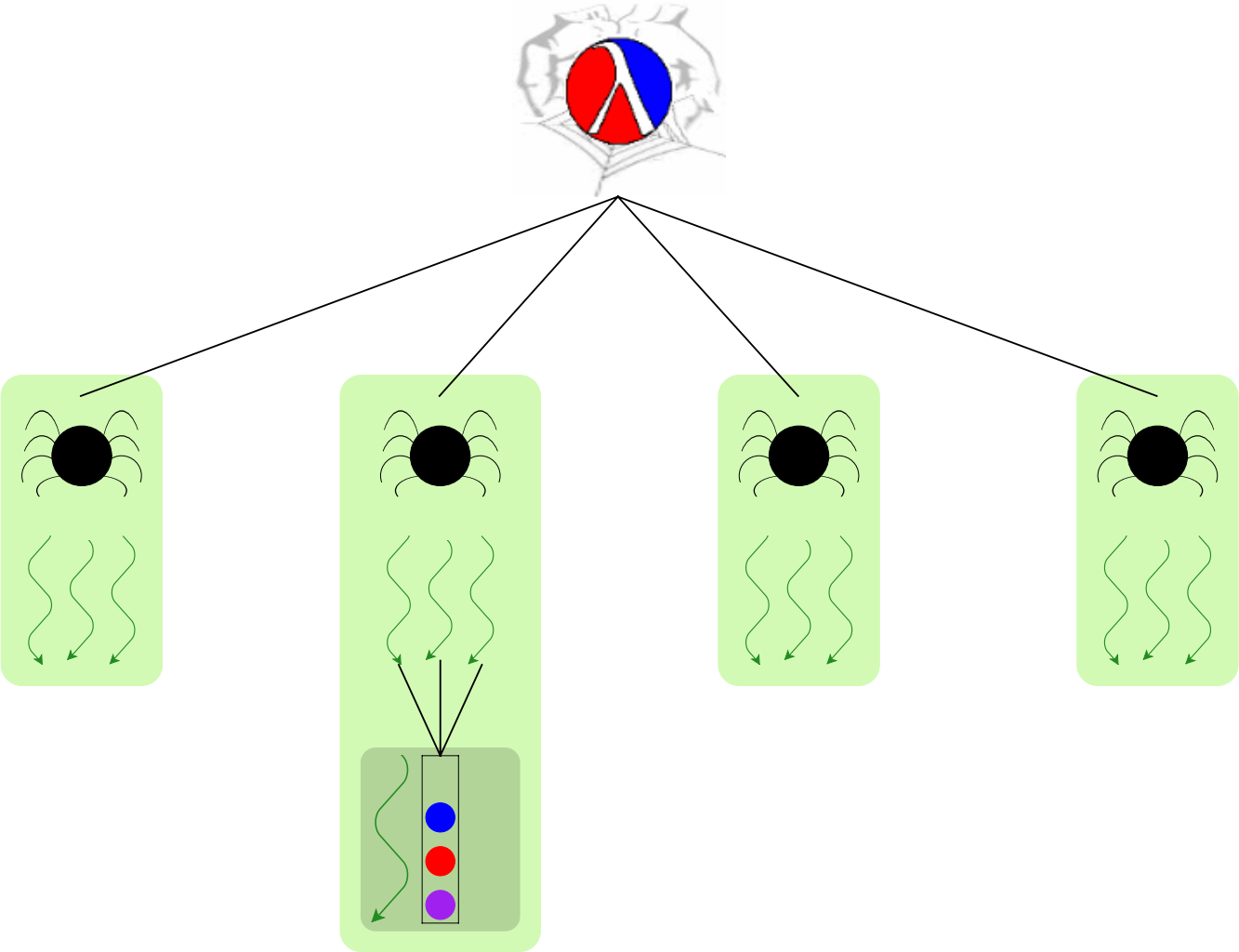
A language to support abstractions:

- Concurrent ML primitives for thread communication
  - Custodians for process hierarchy
- 

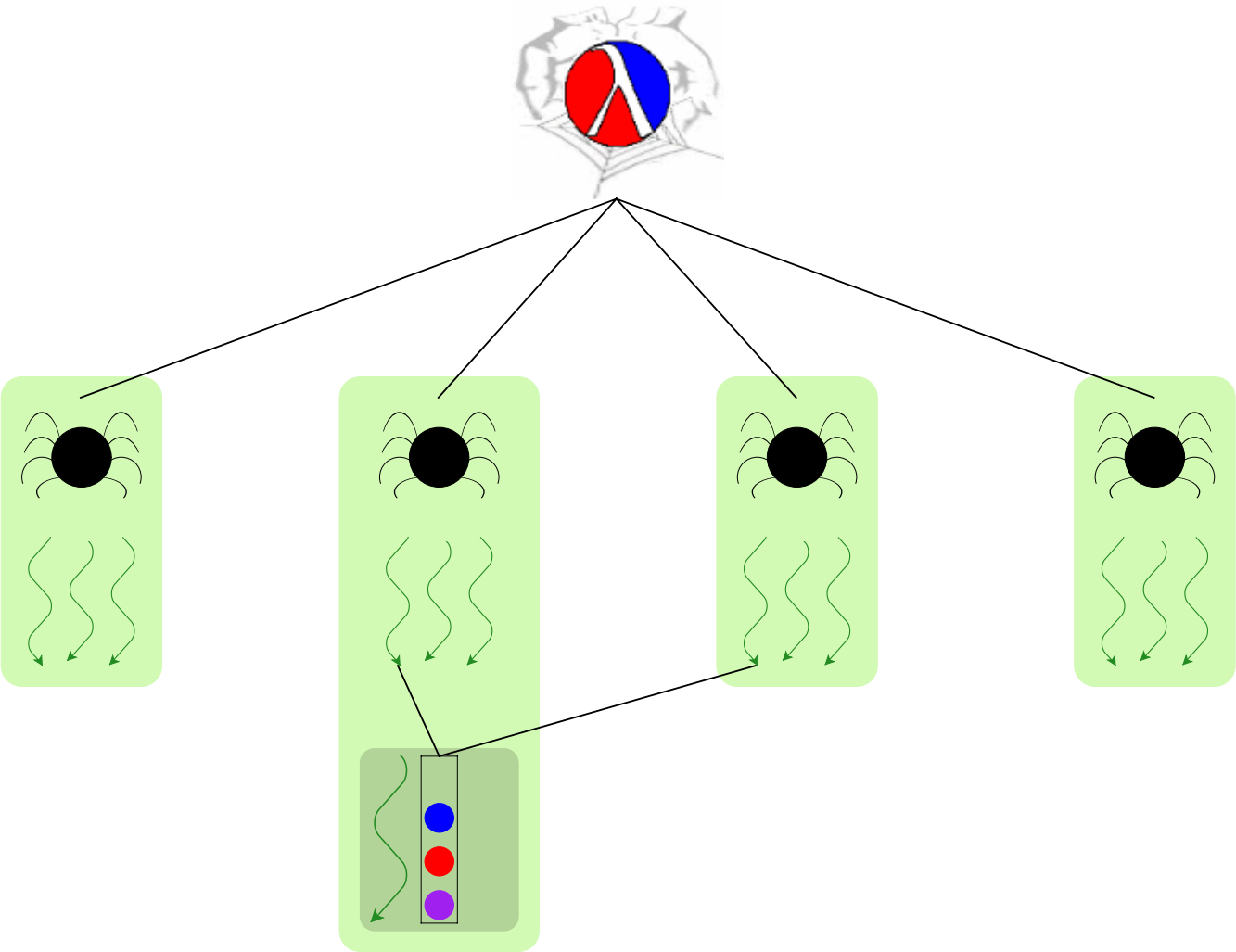
Each abstraction:

- Manager thread for state

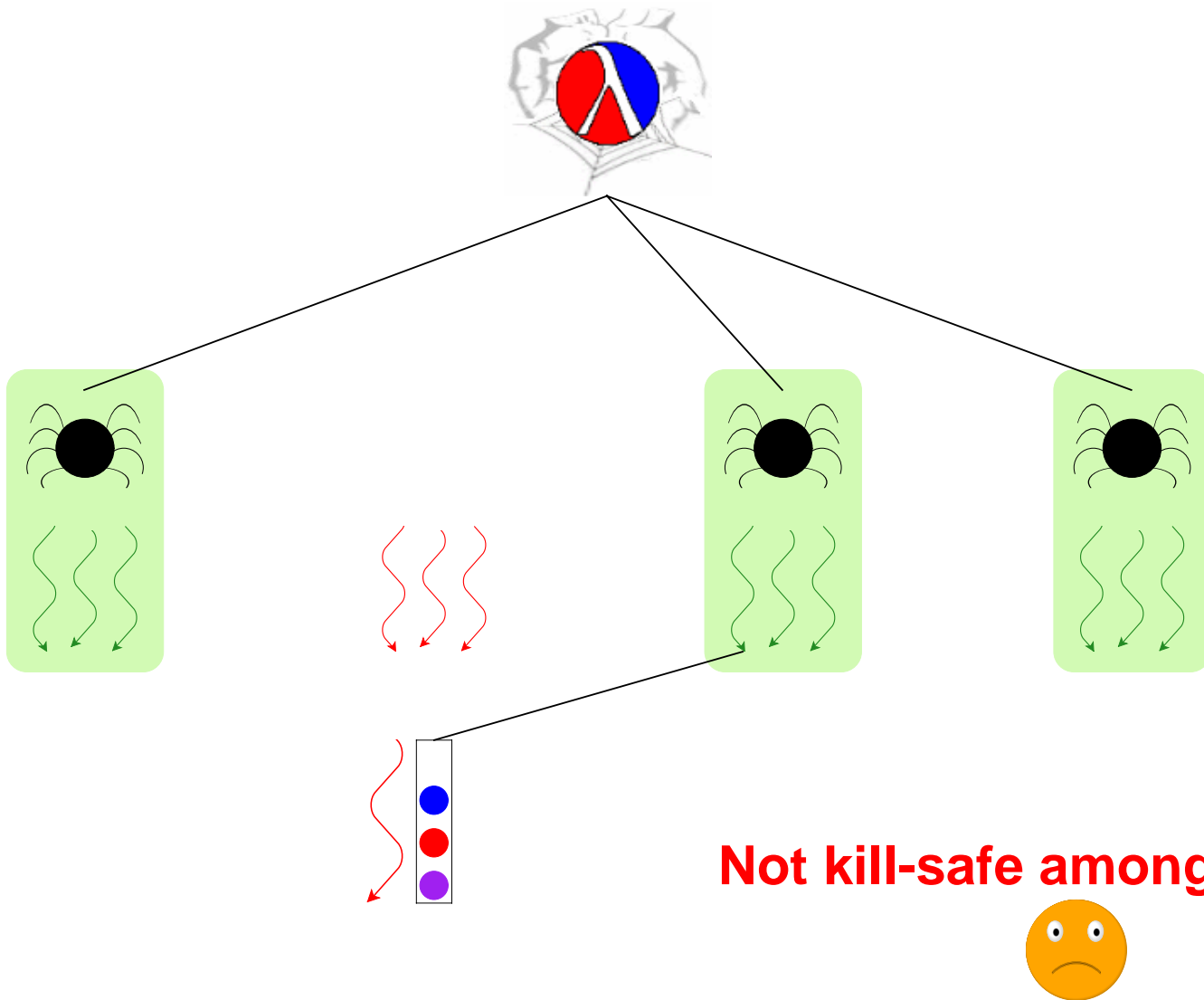
# Towards Kill Safety with Custodians



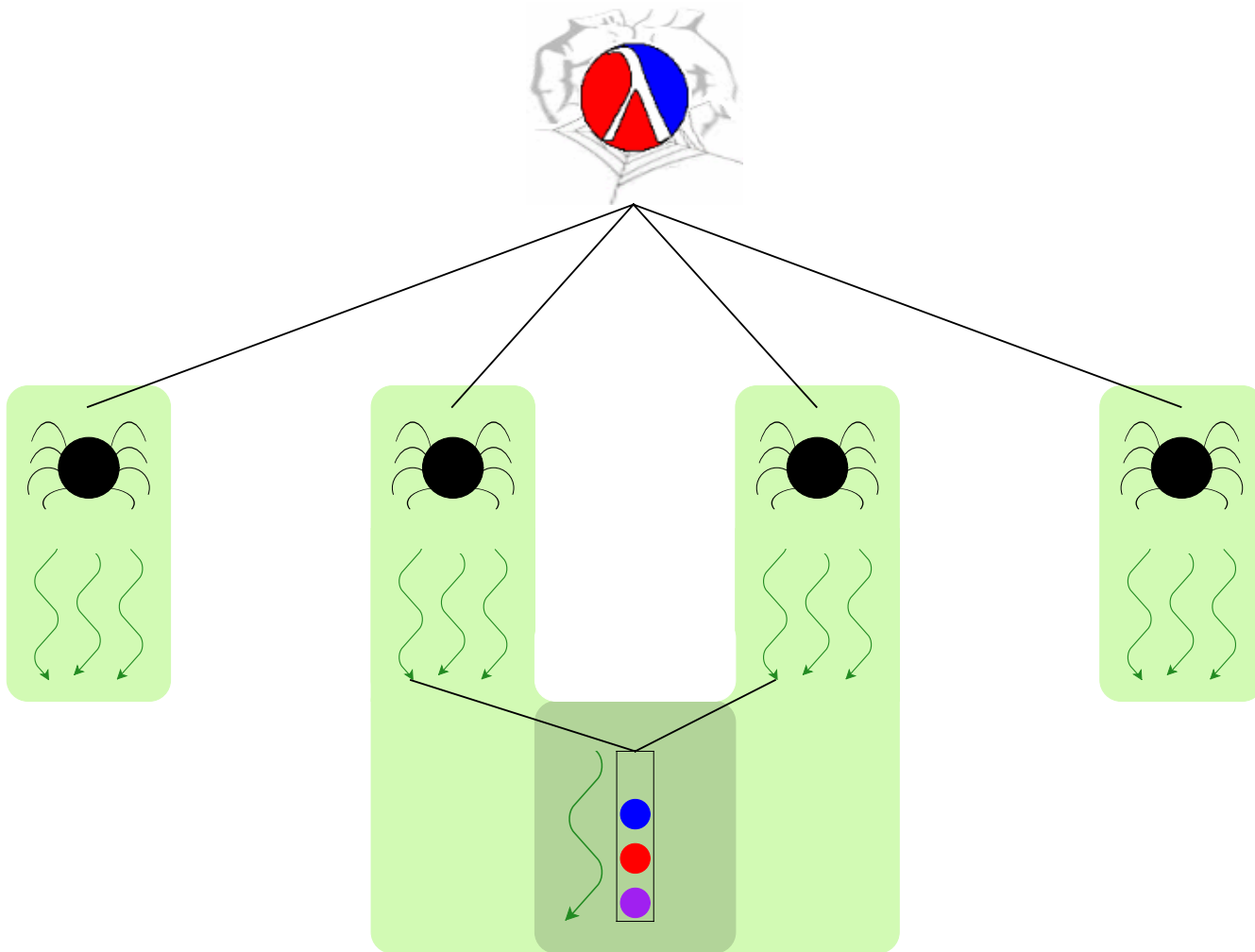
# Towards Kill Safety with Custodians



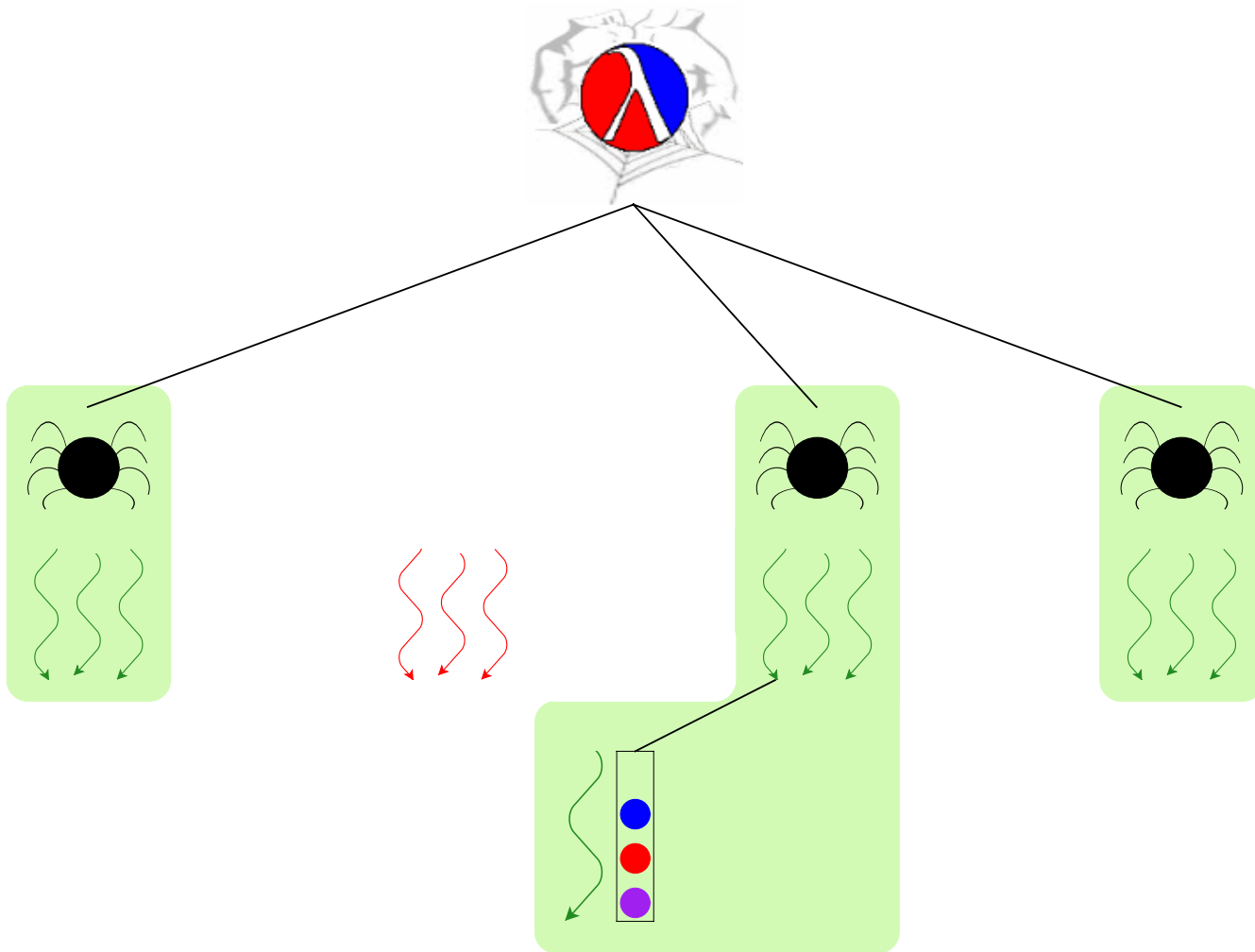
# Towards Kill Safety with Custodians



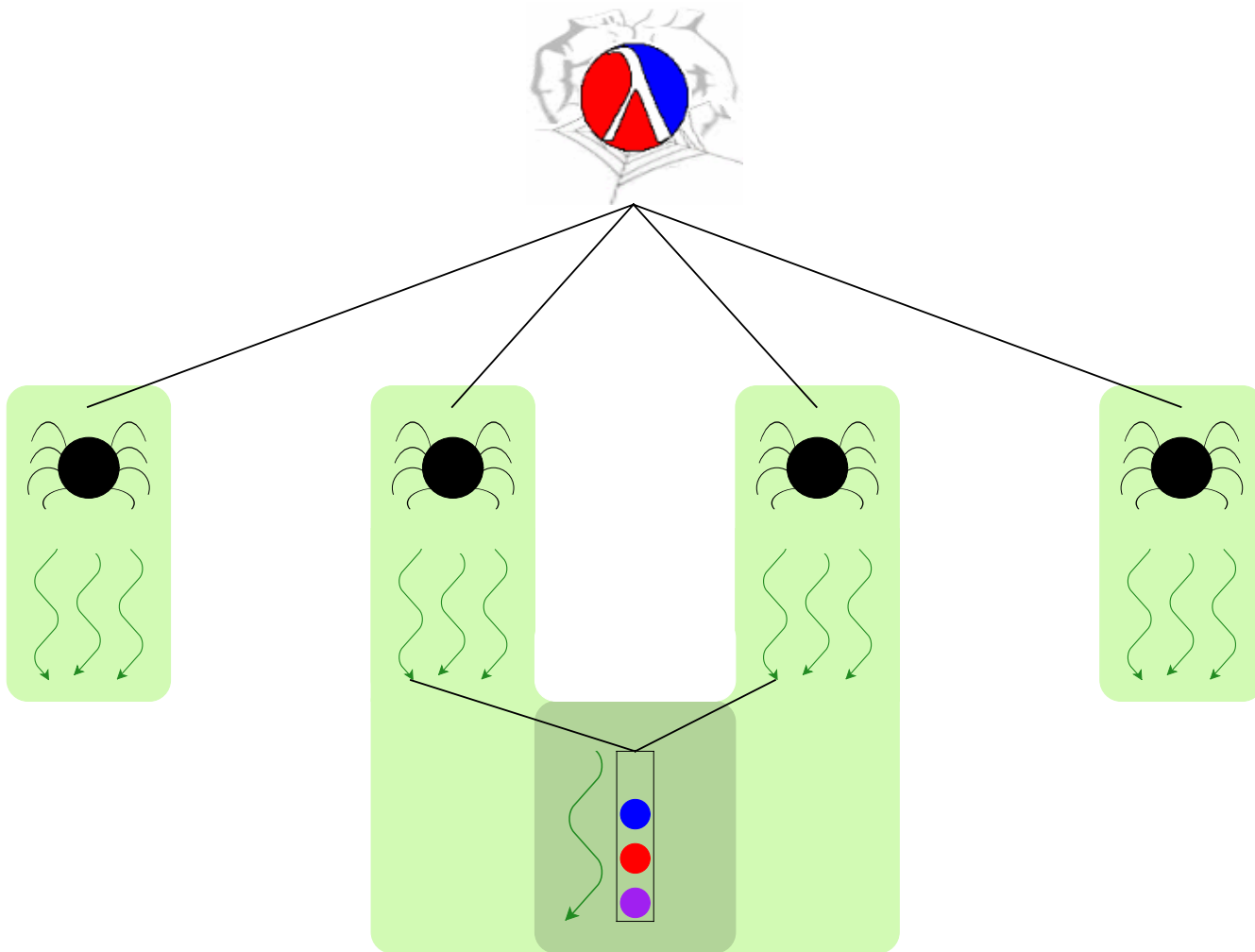
# Kill Safety through Joint Custody



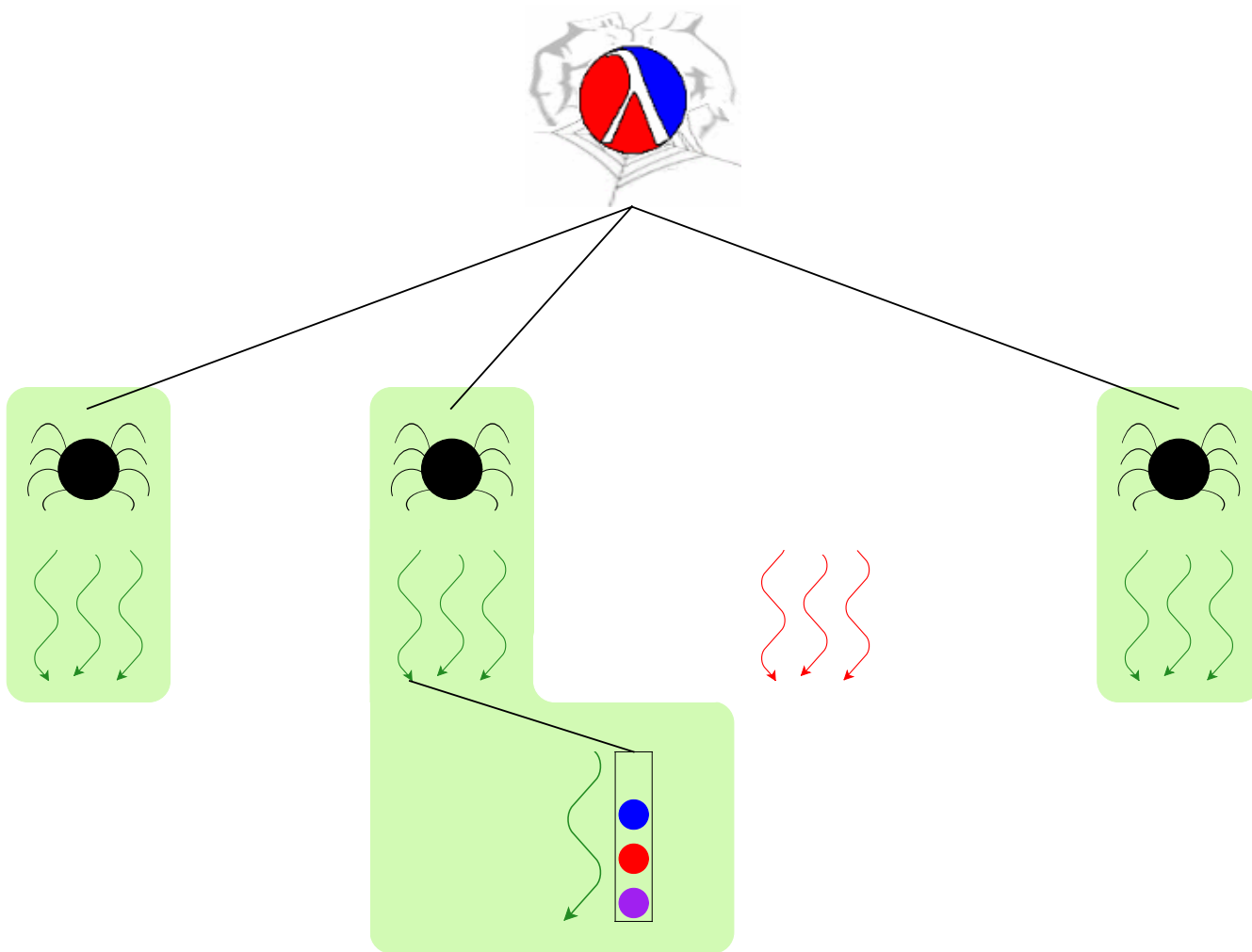
# Kill Safety through Joint Custody



# Kill Safety through Joint Custody

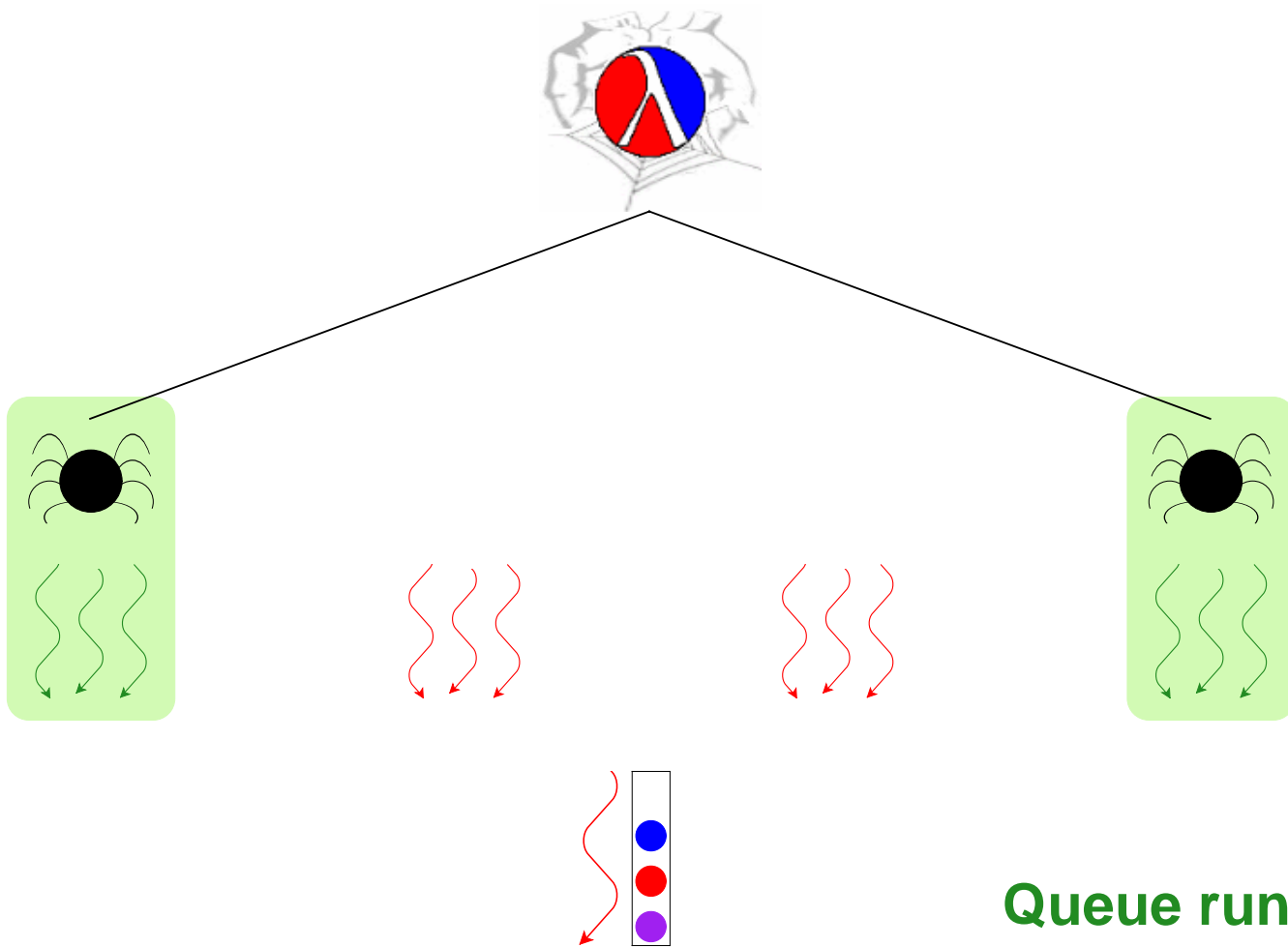


# Kill Safety through Joint Custody





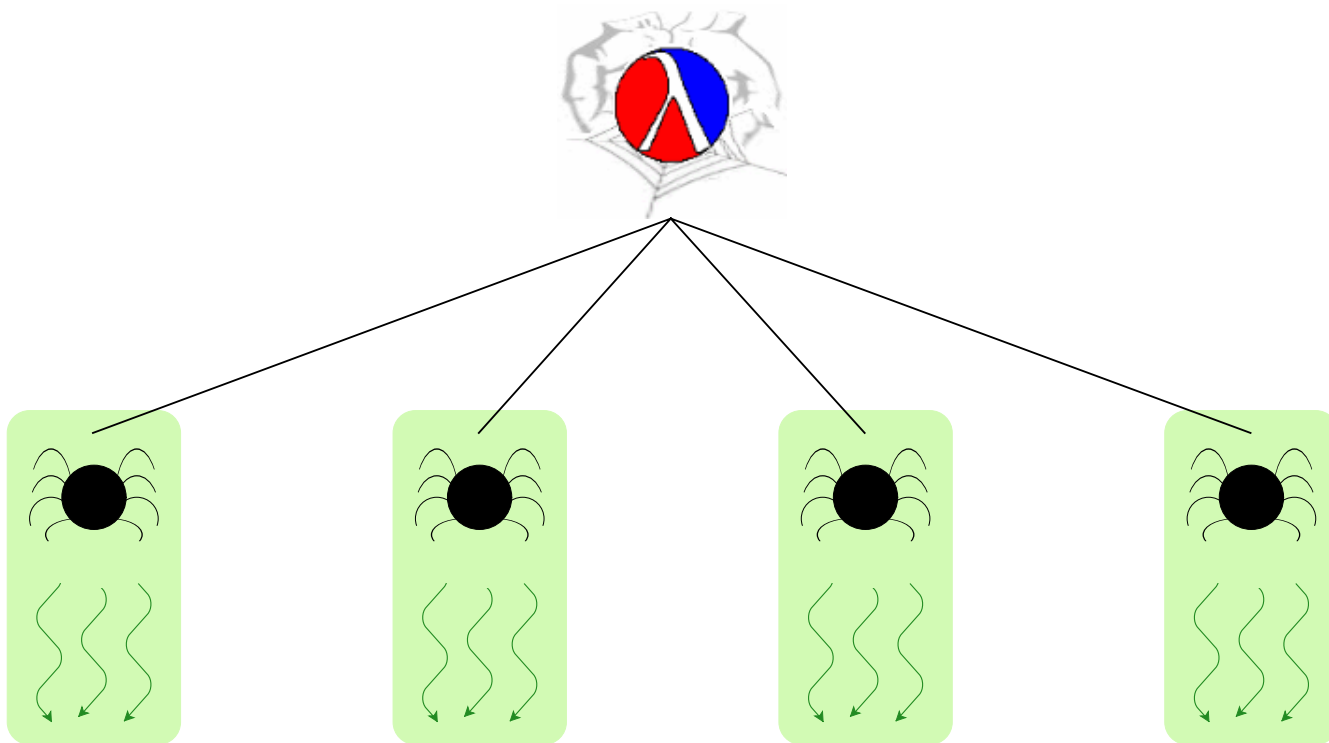
# Kill Safety through Joint Custody



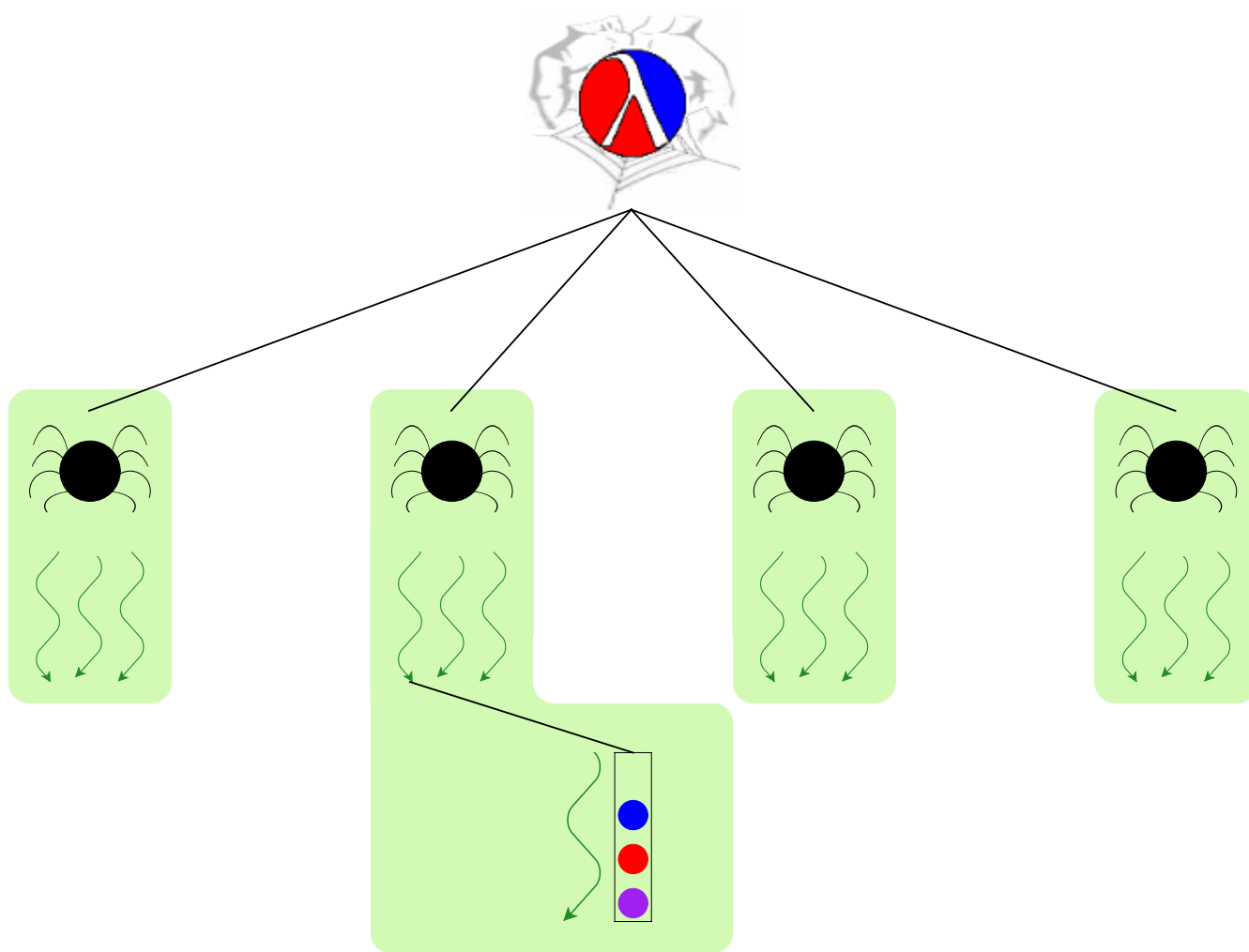
Queue runs exactly  
as long as servlets



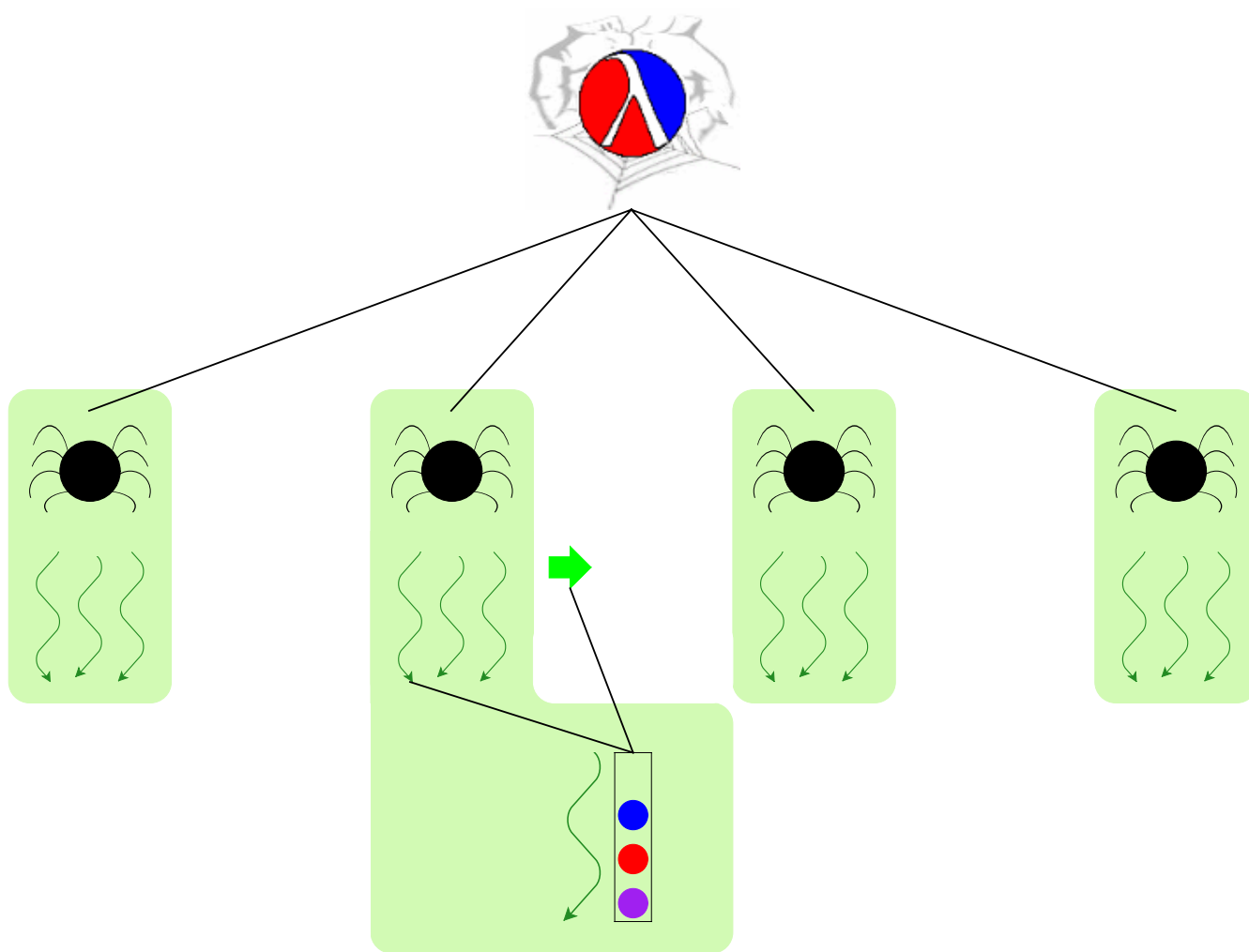
# Why a Thread can have Multiple Custodians



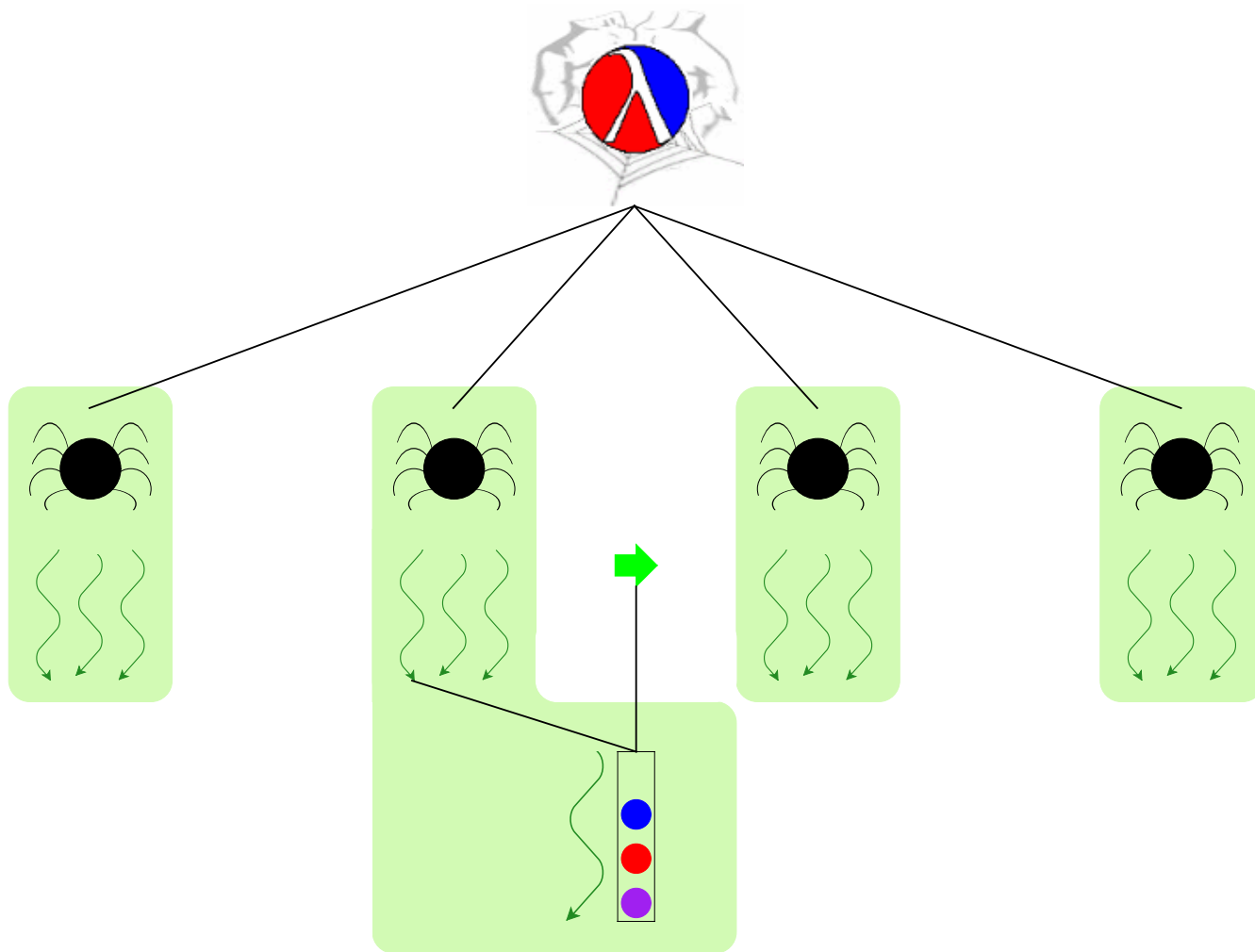
# Why a Thread can have Multiple Custodians



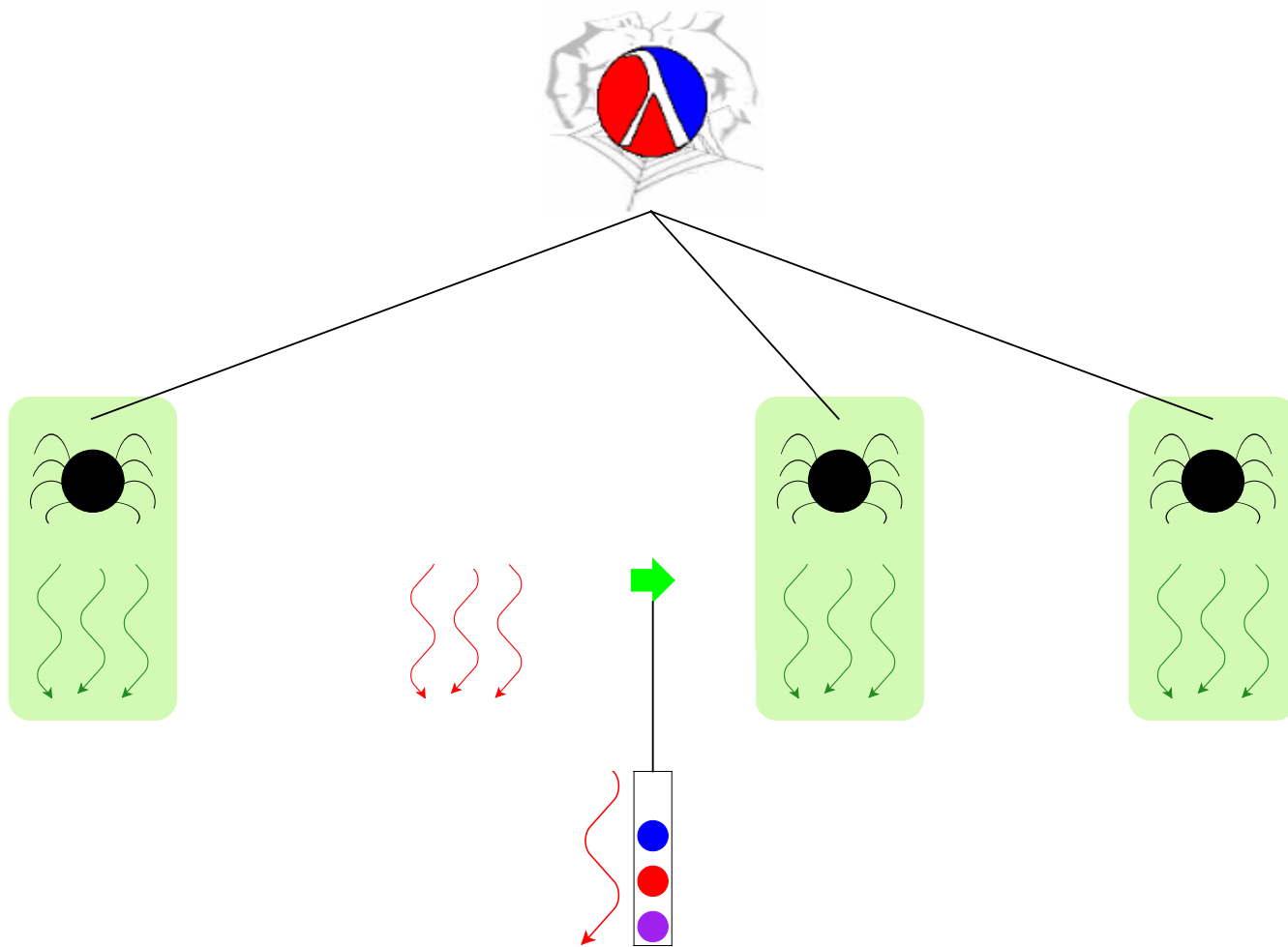
# Why a Thread can have Multiple Custodians



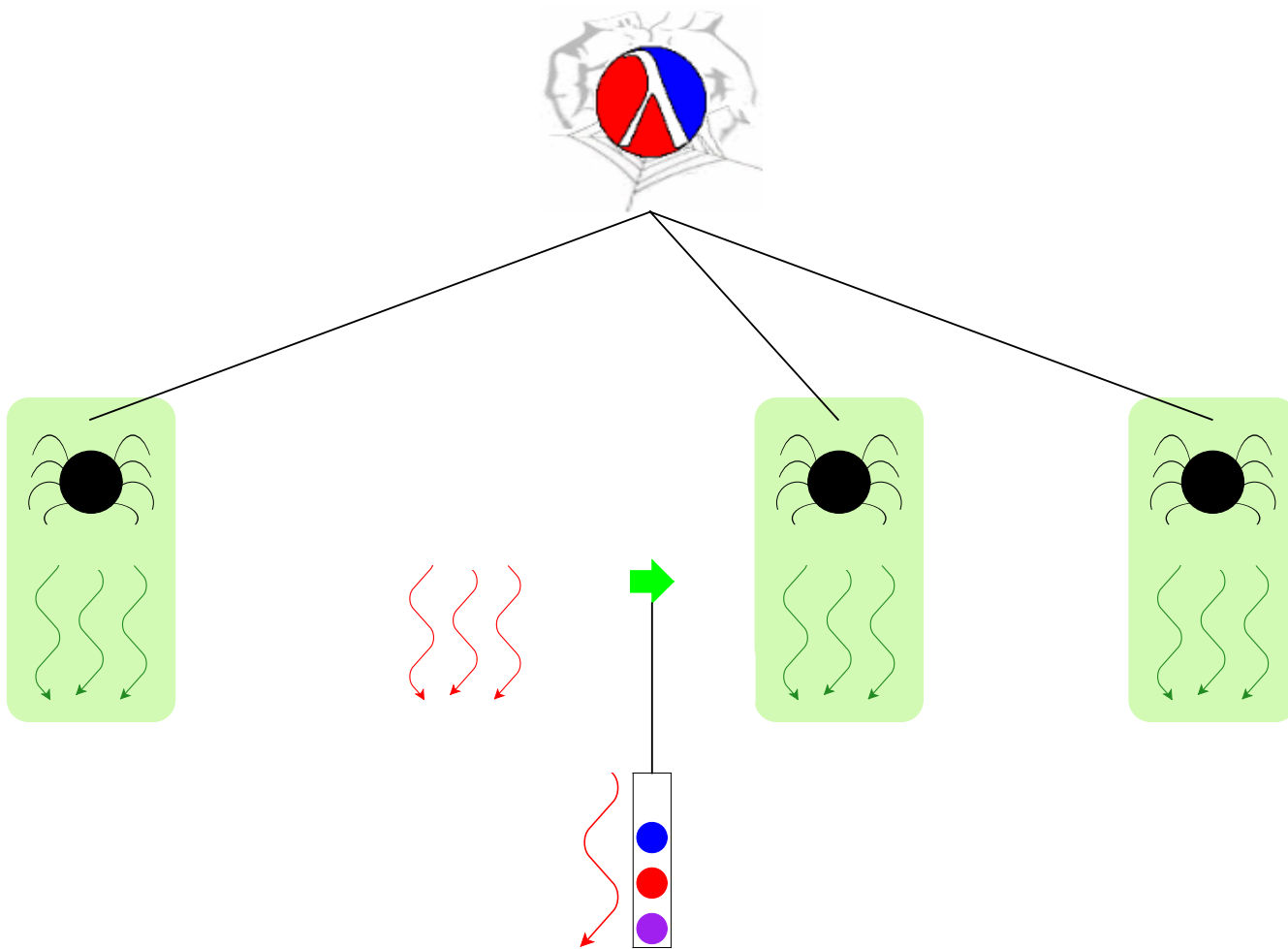
# Why a Thread can have Multiple Custodians



# Why a Thread can have Multiple Custodians

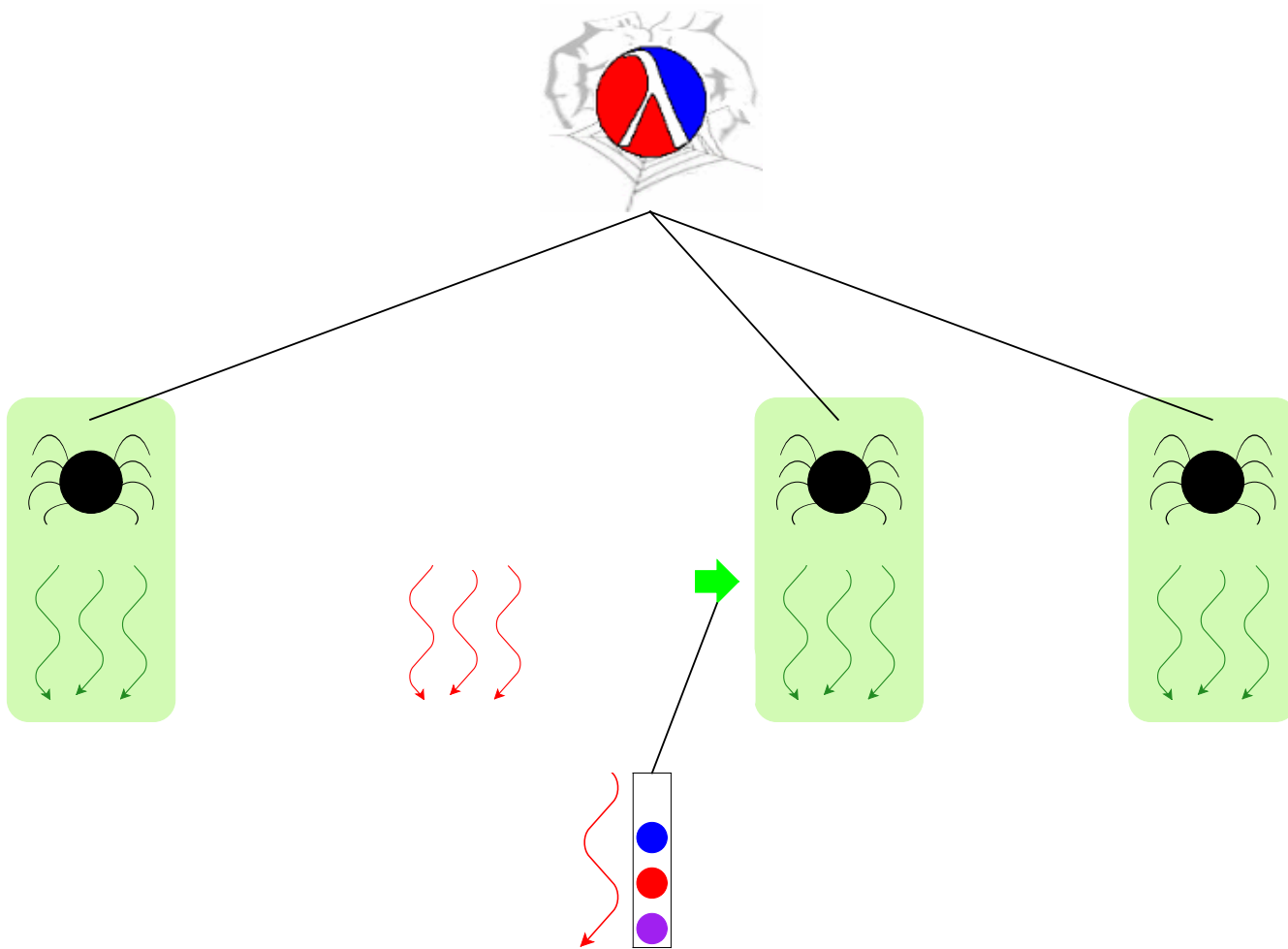


# Why a Thread can have Multiple Custodians



**Queue is only *mostly dead***

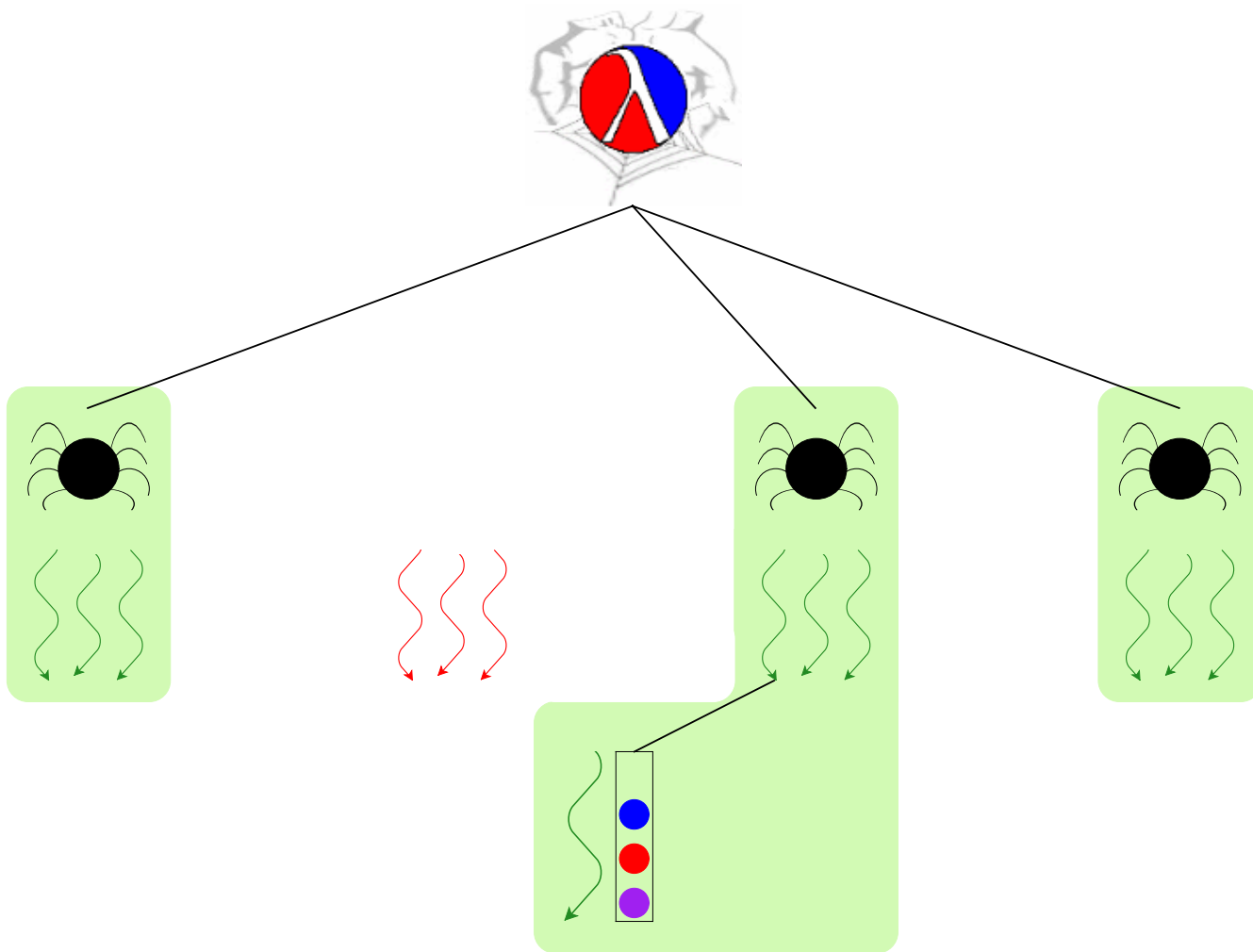
# Why a Thread can have Multiple Custodians



Queue is only *mostly dead*



# Why a Thread can have Multiple Custodians



**Use queue  $\Rightarrow$  grant custodian**

# Kill-Safe Abstractions

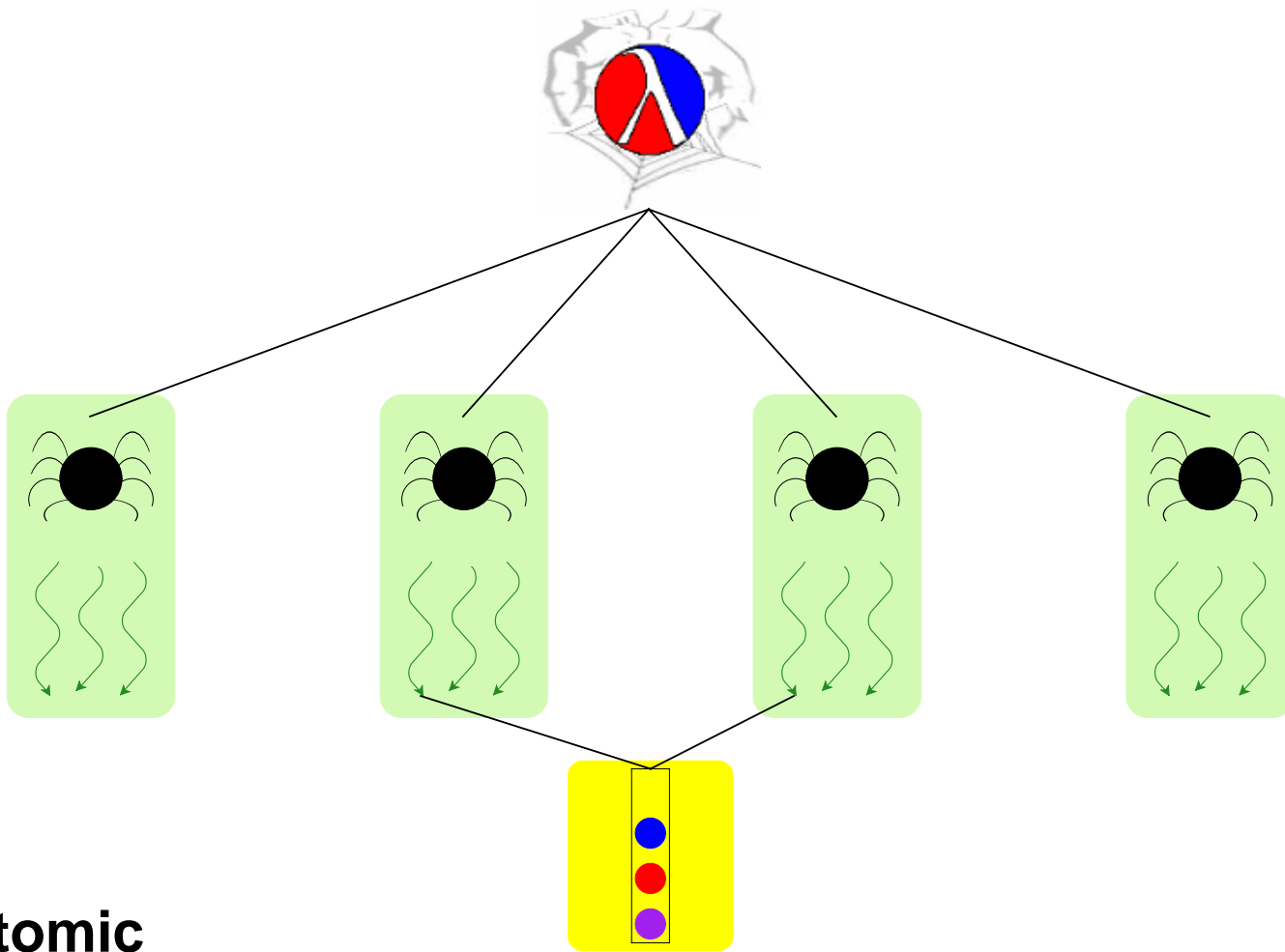
A language to support abstractions:

- Concurrent ML primitives for thread communication
  - Custodians for process hierarchy
  - Operation to grant a thread another custodian
- 

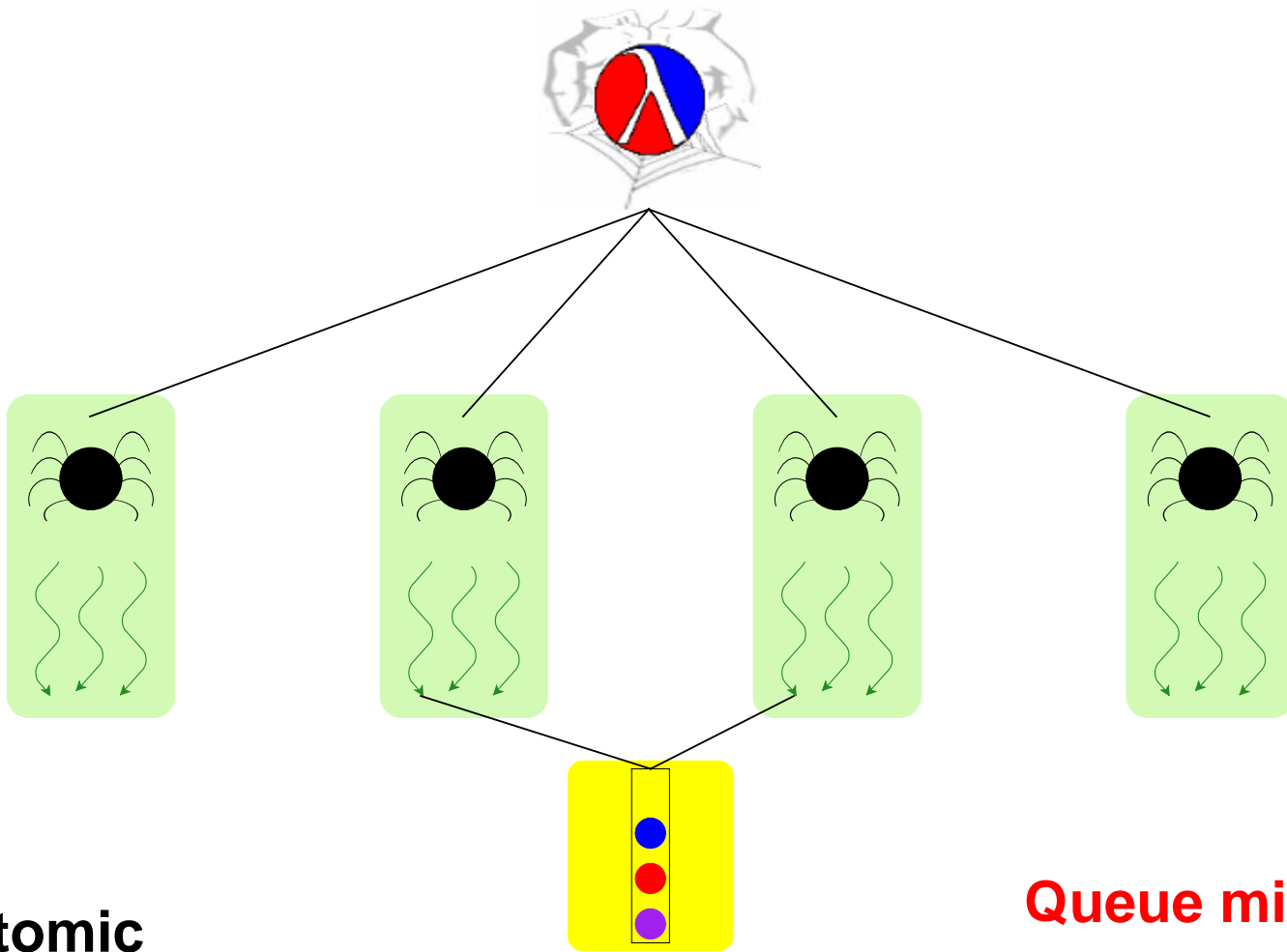
Each abstraction:

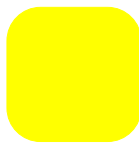
- Manager thread for state
- Each action grants custodian to manager thread

# Non-Solution #1 — Atomic Region



# Non-Solution #1 — Atomic Region

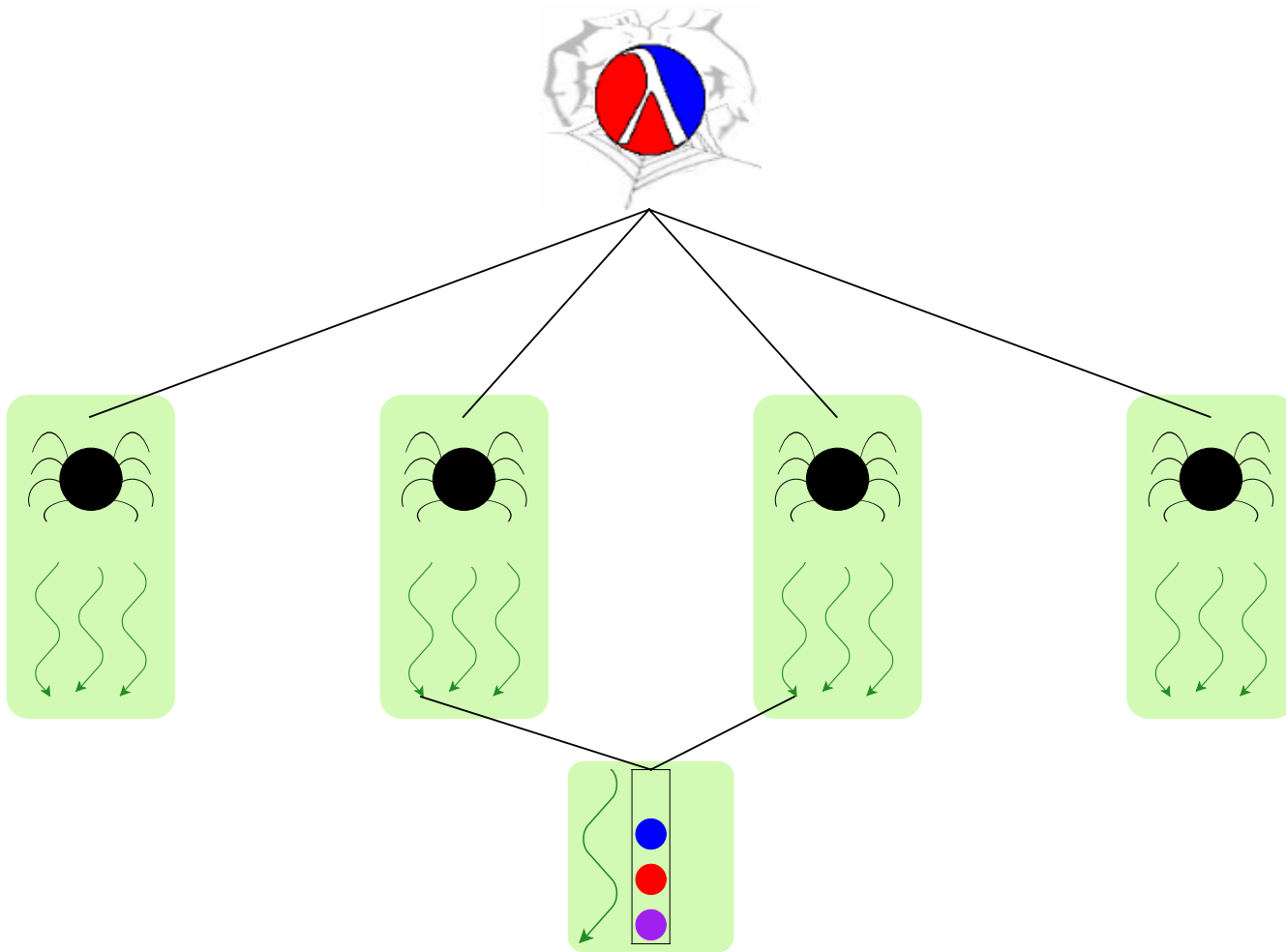


 = atomic

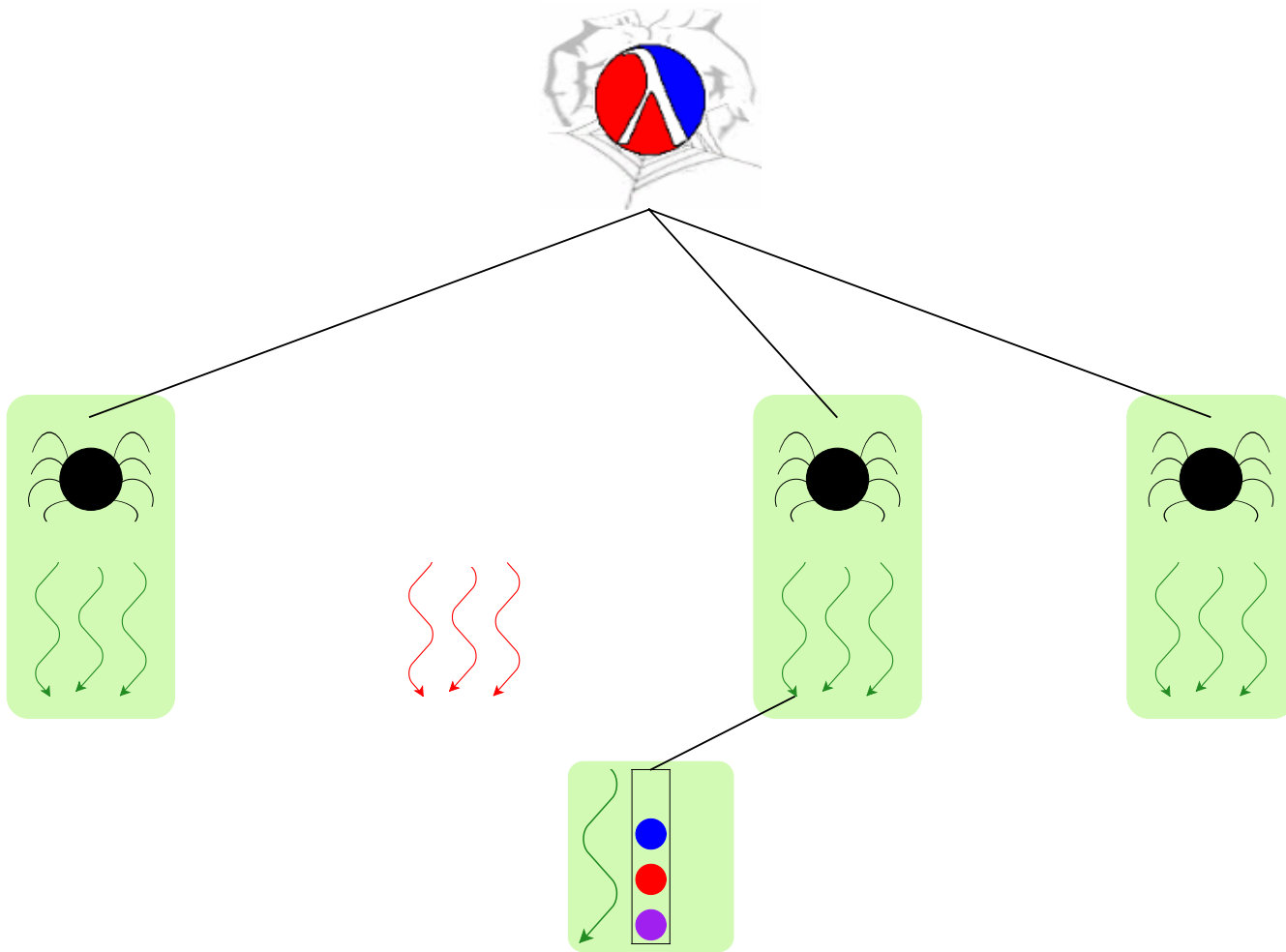
Queue might harm other servlets



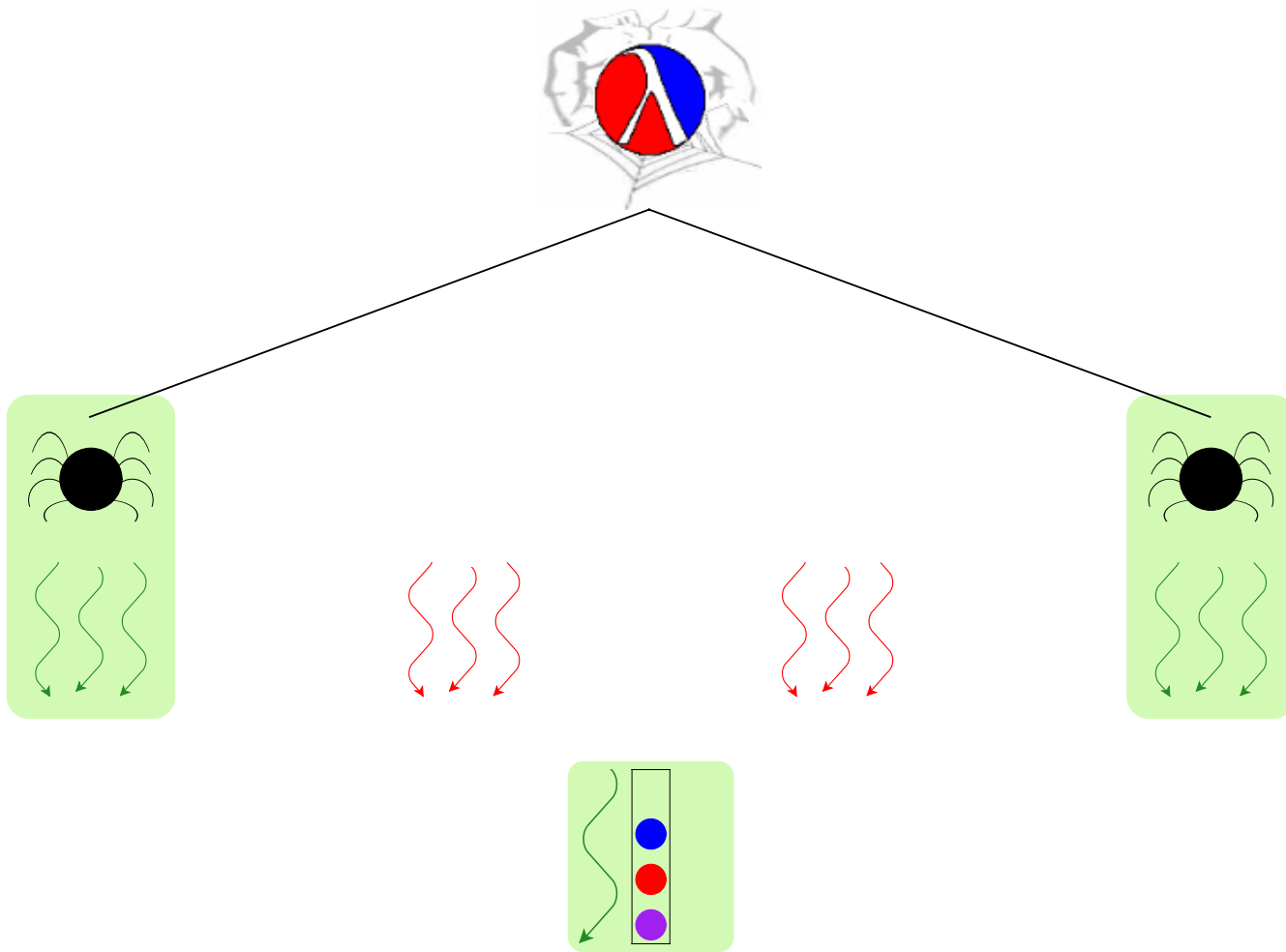
# Non-Solution #2 — Disjoint Process



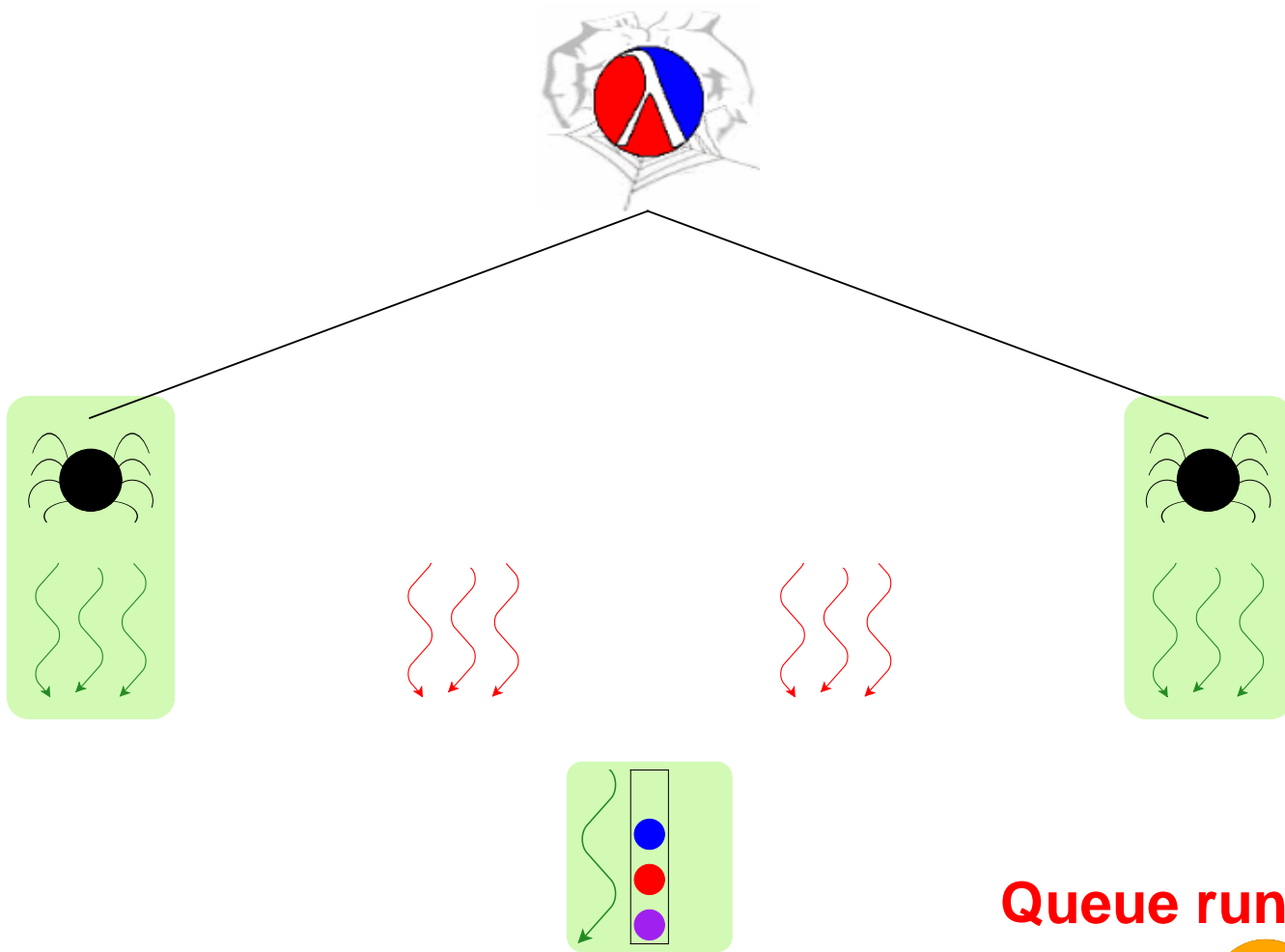
# Non-Solution #2 — Disjoint Process



# Non-Solution #2 — Disjoint Process



# Non-Solution #2 — Disjoint Process

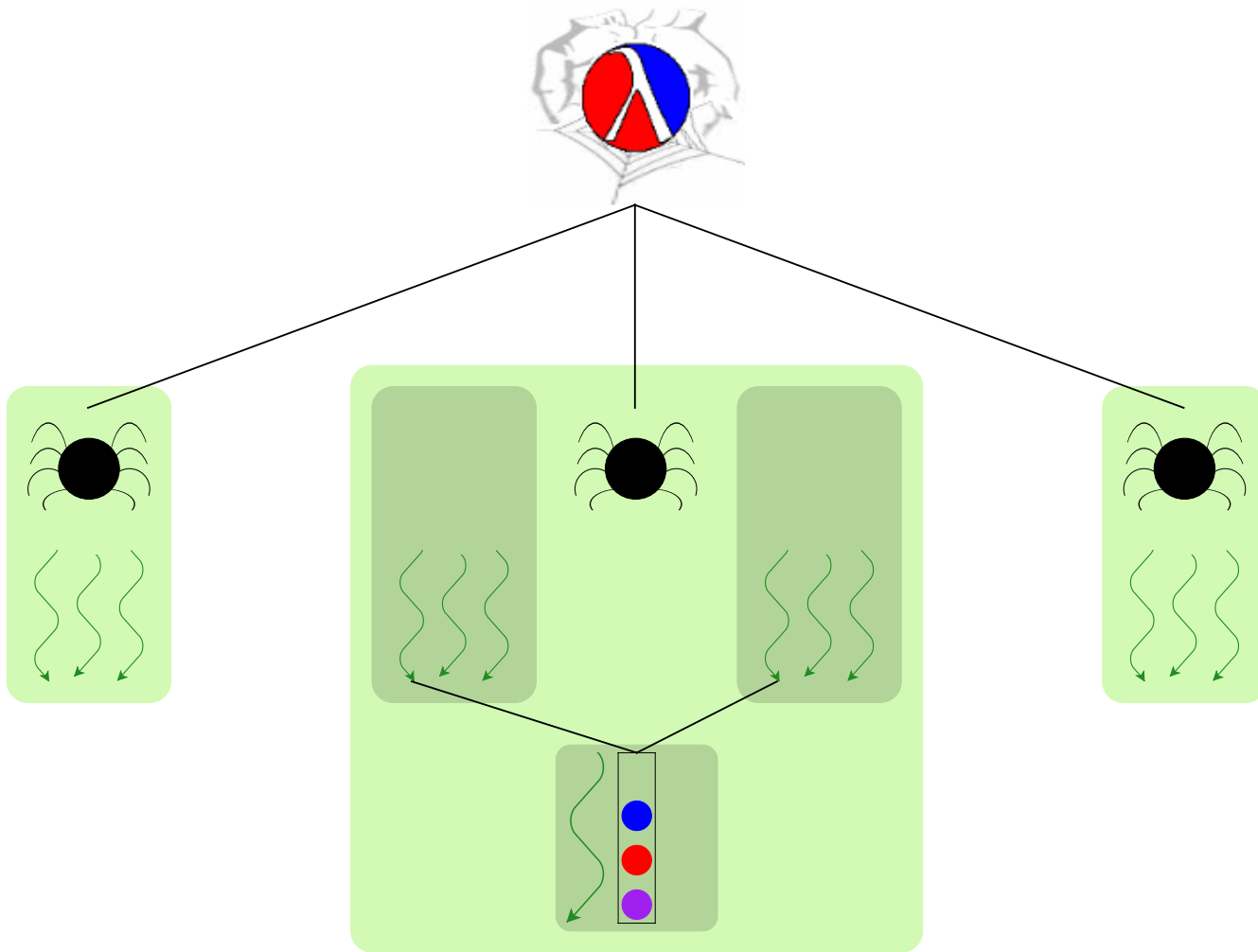


Queue runs forever

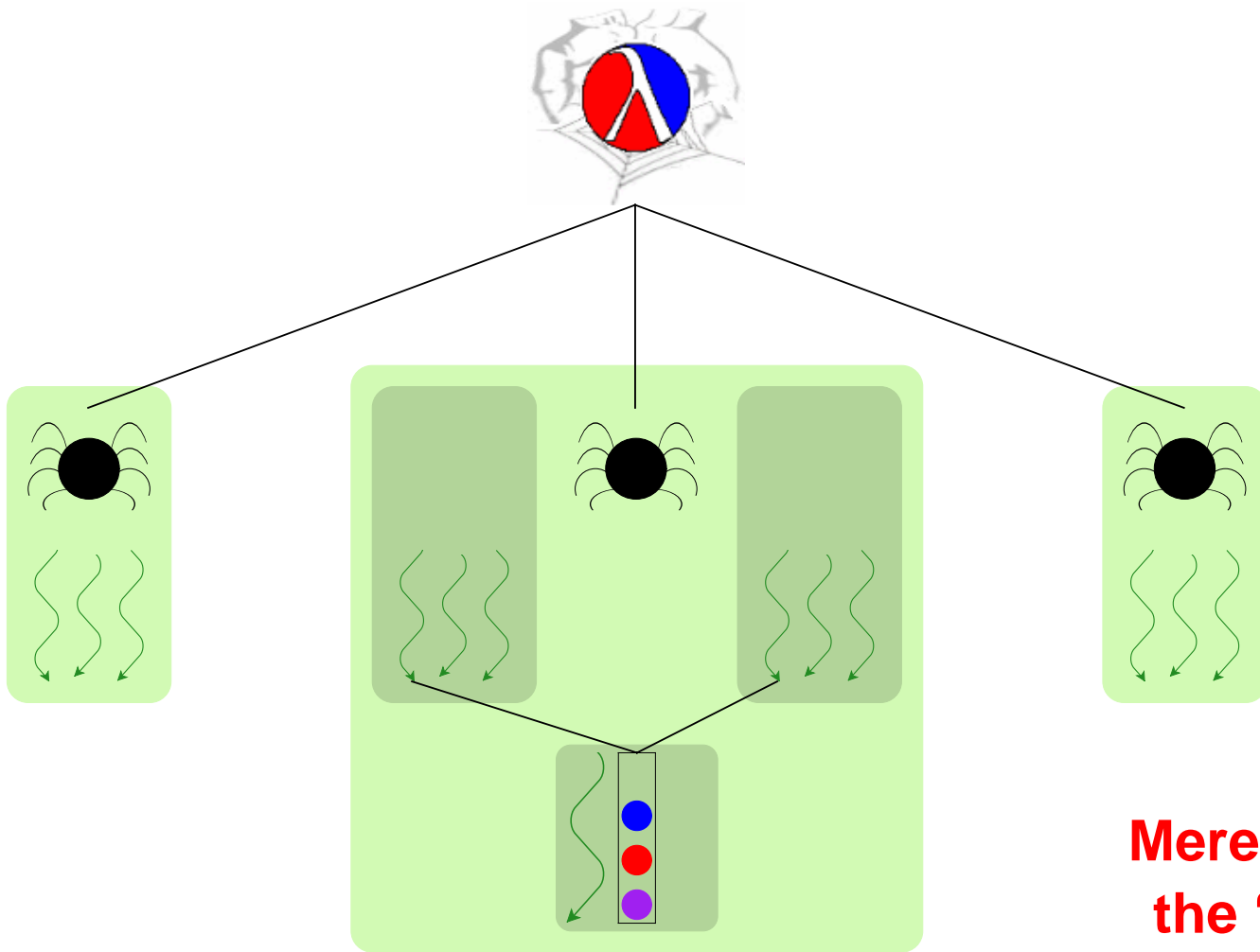




# Non-Solution #3 — Meta-Servlet



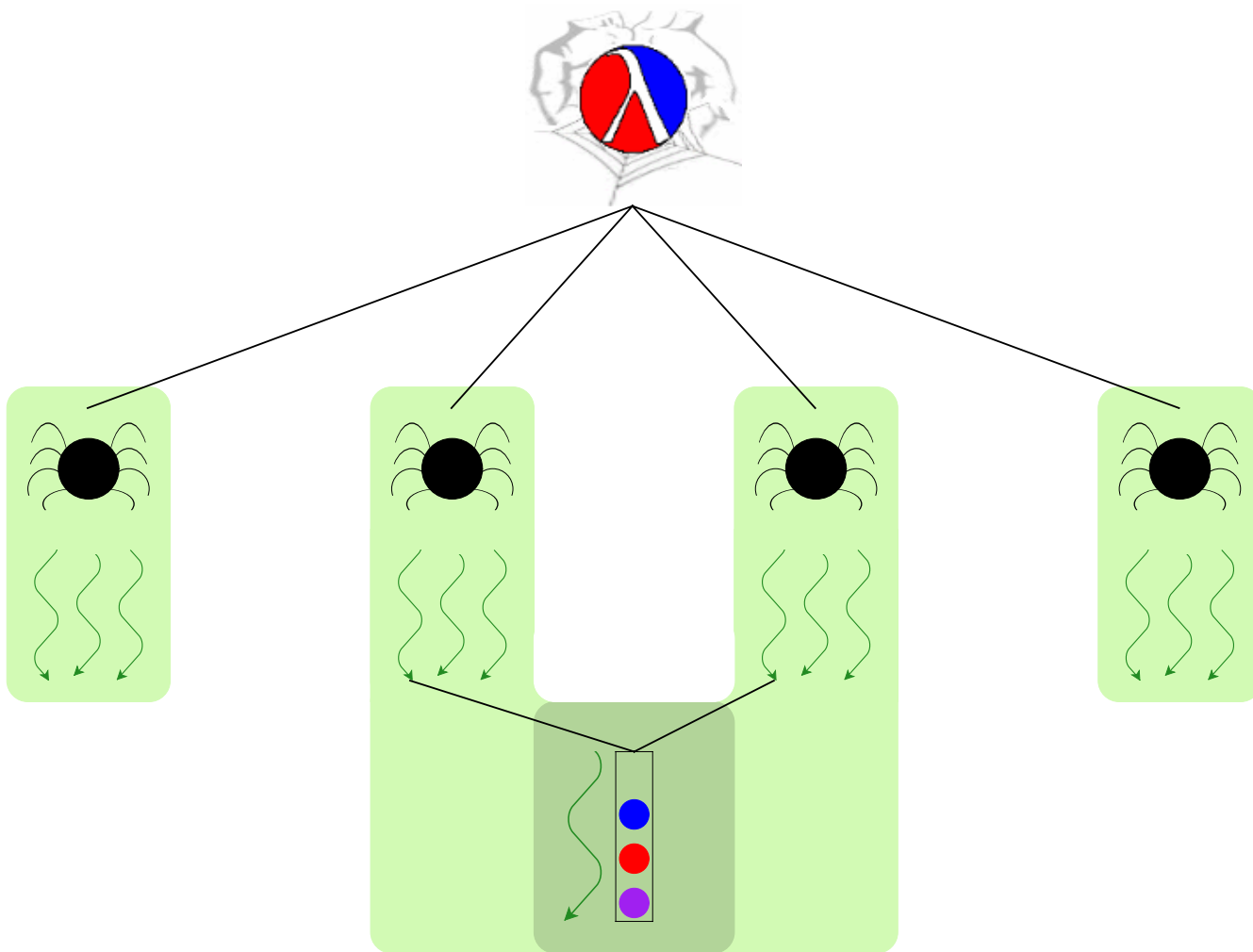
# Non-Solution #3 — Meta-Servlet



**Merely moves  
the “kernel”**



# Solution — Joint Custody



## Details (See Paper)

- Custodians granted through `thread-resume`
- CML's `guard-evt` a natural place for `thread-resume`
- Improved `nack-guard-evt` for two-step protocols
- Kill-safe does not always imply break-safe, nor vice-versa

# A Thread-Safe Queue

```
(define-struct safe-q
  (put-ch get-ch))

(define (safe-queue)
  (define q (queue))
  (define get-ch (channel))
  (define put-ch (channel))
  (define (q-loop)
    (sync
     (choice-evt
      (wrap-evt
       (channel-send get-ch (peek q))
       (lambda () (get q)))
      (wrap-evt
       (channel-recv put-ch)
       (lambda (v) (put q v))))))
    (q-loop))
  (spawn q-loop)
  (make-safe-q put-ch get-ch))

(define (safe-get sq)
  (channel-recv
   (safe-q-get-ch sq)))

(define (safe-put sq v)
  (channel-send
   (safe-q-put-ch sq) v))
```

# A Kill-Safe Queue

```
(define-struct safe-q
  (manager-t put-ch get-ch))

(define (safe-queue)
  (define q (queue))
  (define get-ch (channel))
  (define put-ch (channel))
  (define (q-loop)
    (sync
     (choice-evt
      (wrap-evt
       (channel-send get-ch (peek q))
       (lambda () (get q)))
      (wrap-evt
       (channel-recv put-ch)
       (lambda (v) (put q v))))))
    (q-loop))
  (define manager-t (spawn q-loop))
  (make-safe-q manager-t put-ch get-ch))
```

```
(define (safe-get sq)
  (resume sq)
  (channel-recv
   (safe-q-get-ch sq)))

(define (safe-put sq v)
  (resume sq)
  (channel-send
   (safe-q-put-ch sq) v))

(define (resume sq)
  (thread-resume
   (safe-q-manager-t sq)
   (current-thread)))
```