Parallelization of Abstracted Abstract Machines

Leif Andersen

Matthew Might

University of Utah

- What is Control Flow Analysis?

- What is Control Flow Analysis?
- What is Abstract Interpretation?

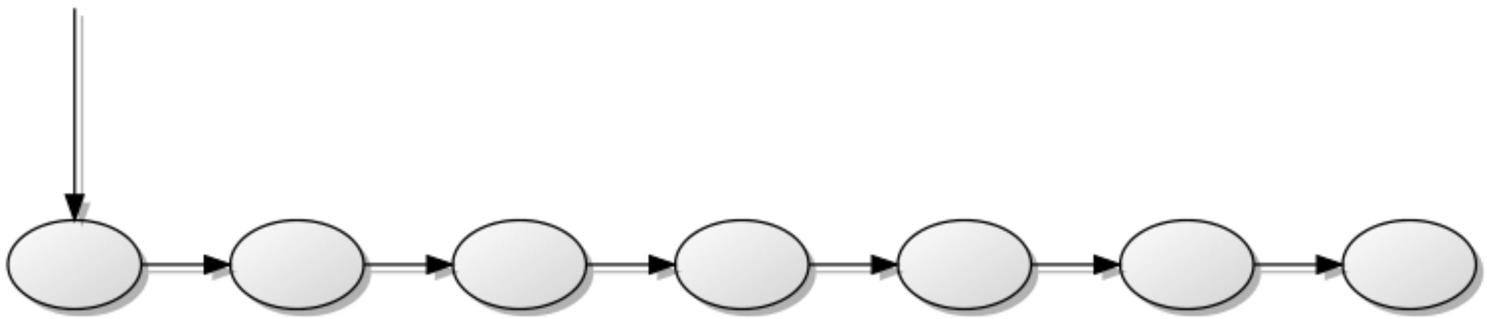
- What is Control Flow Analysis?
- What is Abstract Interpretation?
- How can Control Flow Analysis be parallelized?

- What is Control Flow Analysis?
- What is Abstract Interpretation?
- How can Control Flow Analysis be parallelized? (Using Scala)

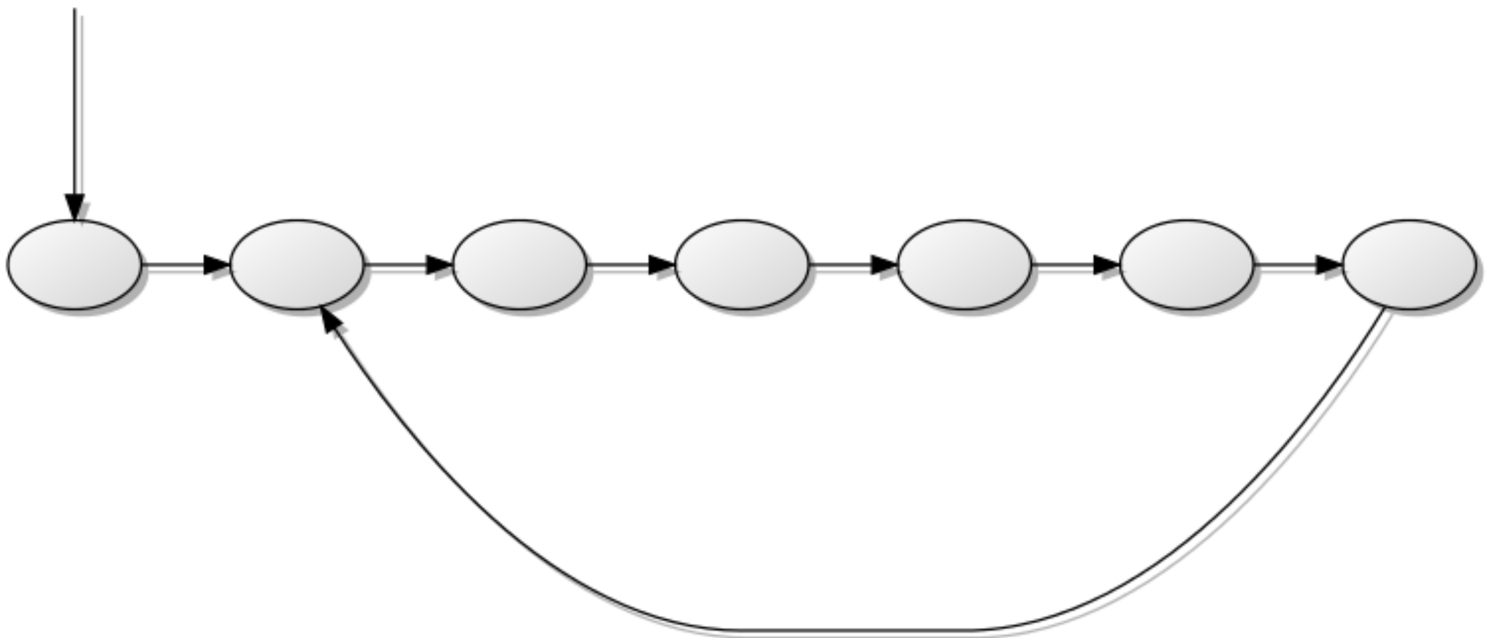
What is Control Flow Analysis?

```
(define (output t)
  (if (token? t)
      (let ([n (token-name t)]
            [v (token-value t)])
        (match n
          ['KEYWORD (display (format "(~a ~a)\n" n v))]
          ['LIT (display (format "(~a ~a)\n" n v))]
          [else (display (format "(~a \"~a\")\n" n v))]))
      (display (format "(~a)\n" t))))
(define (call-lexer lexer ς port)
  (let ([ς* (lexer ς port)])
    (when ς* (call-lexer lexer ς* port))))
(call-lexer lexer-code 'start
            (input-port-append
             #f (open-input-string "\n")
             (current-input-port)))
```

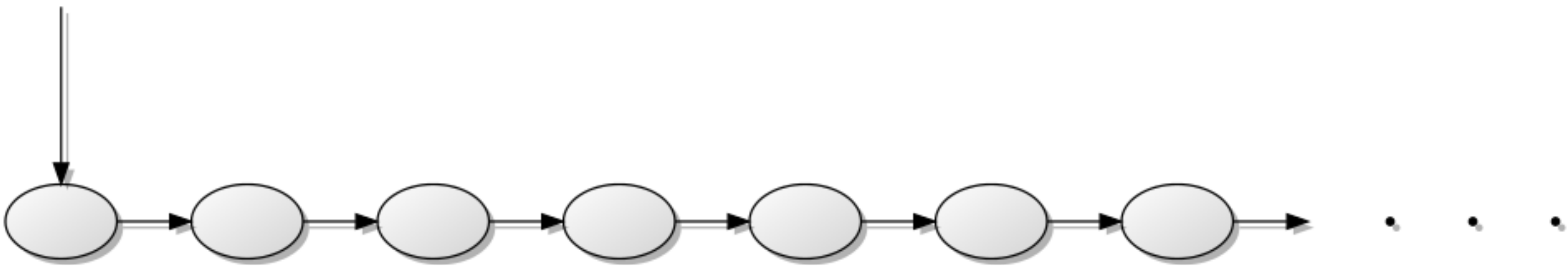
```
(define (output t)
  (if (token? t)
      (let ([n (token-name t)]
            [v (token-value t)])
        (match n
          ['KEYWORD (display (format "(~a ~a)\n" n v))]
          ['LIT (display (format "(~a ~a)\n" n v))]
          [else (display (format "(~a \"~a\")\n" n v))]))
      (display (format "(~a)\n" t))))
(define (call-lexer lexer ς port)
  (let ([ς* (lexer ς port)])
    (when ς* (call-lexer lexer ς* port))))
(call-lexer lexer-code 'start
            (input-port-append
             #f (open-input-string "\n")
             (current-input-port)))
```



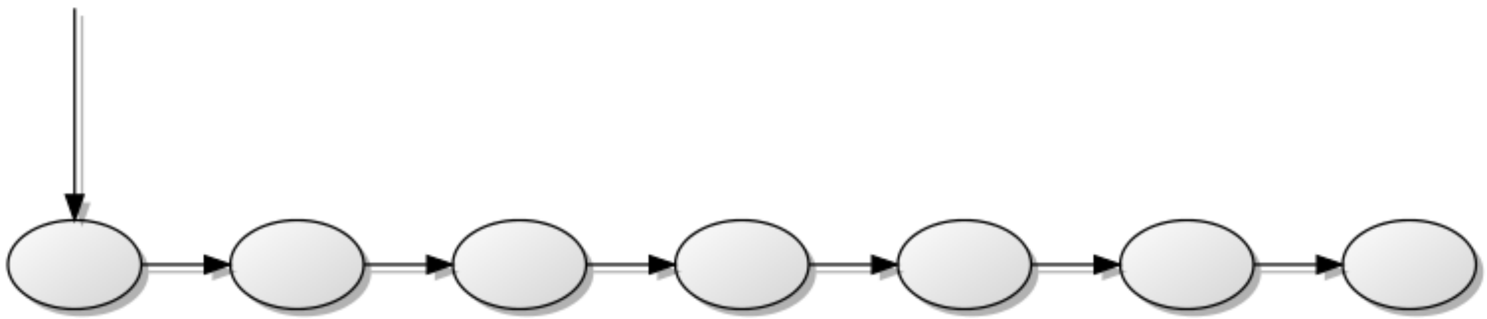
```
(define (output t)
  (if (token? t)
      (let ([n (token-name t)]
            [v (token-value t)])
        (match n
          ['KEYWORD (display (format "(~a ~a)\n" n v))]
          ['LIT (display (format "(~a ~a)\n" n v))]
          [else (display (format "(~a \"~a\")\n" n v))]))
      (display (format "(~a)\n" t))))
(define (call-lexer lexer ς port)
  (let ([ς* (lexer ς port)])
    (when ς* (call-lexer lexer ς* port))))
(call-lexer lexer-code 'start
            (input-port-append
             #f (open-input-string "\n")
             (current-input-port)))
```

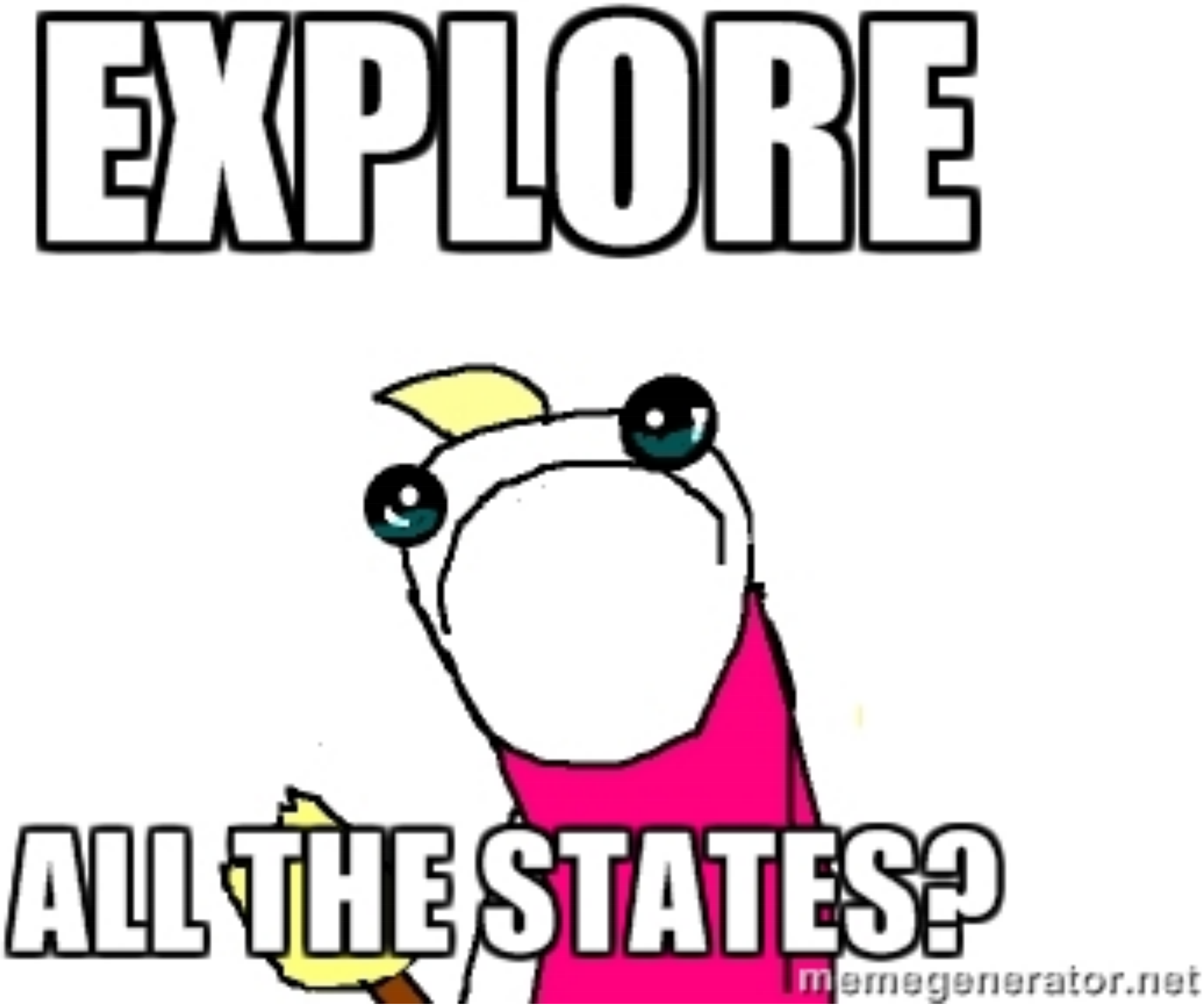


```
(define (output t)
  (if (token? t)
      (let ([n (token-name t)]
            [v (token-value t)])
        (match n
          ['KEYWORD (display (format "(~a ~a)\n" n v))]
          ['LIT (display (format "(~a ~a)\n" n v))]
          [else (display (format "(~a \"~a\")\n" n v))]))
      (display (format "(~a)\n" t))))
(define (call-lexer lexer ς port)
  (let ([ς* (lexer ς port)])
    (when ς* (call-lexer lexer ς* port))))
(call-lexer lexer-code 'start
            (input-port-append
             #f (open-input-string "\n")
             (current-input-port)))
```



```
(define (output t)
  (if (token? t)
      (let ([n (token-name t)]
            [v (token-value t)])
        (match n
          ['KEYWORD (display (format "(~a ~a)\n" n v))]
          ['LIT (display (format "(~a ~a)\n" n v))]
          [else (display (format "(~a \"~a\")\n" n v))]))
      (display (format "(~a)\n" t))))
(define (call-lexer lexer ς port)
  (let ([ς* (lexer ς port)])
    (when ς* (call-lexer lexer ς* port))))
(call-lexer lexer-code 'start
            (input-port-append
             #f (open-input-string "\n")
             (current-input-port)))
```




EXPLORE
ALL THE STATES?
memegenerator.net

CESK

Control Environment Store Kontinuation

CESK

CESK

Control

Expression

Expression Environment

Expression Registers

Expression
Registers
Store

Expression
Registers
Heap

Expression
Registers
Heap
Kontinuation

Expression
Registers
Heap
Stack

```scheme
(define (eval e env)
  (cond
    ((symbol? e)        (cadr (assq e env)))
    ((eq? (car e) 'λ)  (cons e env))
    (else               (apply (eval (car e) env)
                               (eval (cadr e) env)))))
(define (apply f x)
  (eval (cddr (car f)) (cons (list (cadr (car f)) x)
                             (cdr f))))
(display (eval (read) '())) (newline)
```

CE

CEK

Environment

Environment

$$\text{VAR} \rightarrow CLO + \mathbf{halt}$$

Where:

$$CLO = \mathbf{Lambda} \times Env$$

CEK

CESK

Environment

$$\rho : \mathsf{VAR} \to Addr$$

Store

$$\sigma : Addr \to CLO + \mathbf{halt}$$

CESK


e,ρ,σ,κ

CESK

CES

Continuation Passing Style

```
<cexp> ::= (<aexp> <aexp>*)
<aexp> ::= (λ (<var>*) <cexp>)
        |  <var>
        |  <halt>
```

$$(\lambda \quad (x) \quad x)$$

$(\lambda\ \cancel{(x)}\ x)$

```
(λ   (x  k)
     (k  x))
```

Environment

$$\rho : \mathsf{VAR} \rightarrow Addr$$

Environment

$$\rho : \mathsf{VAR} \to \widehat{Addr}$$

Store

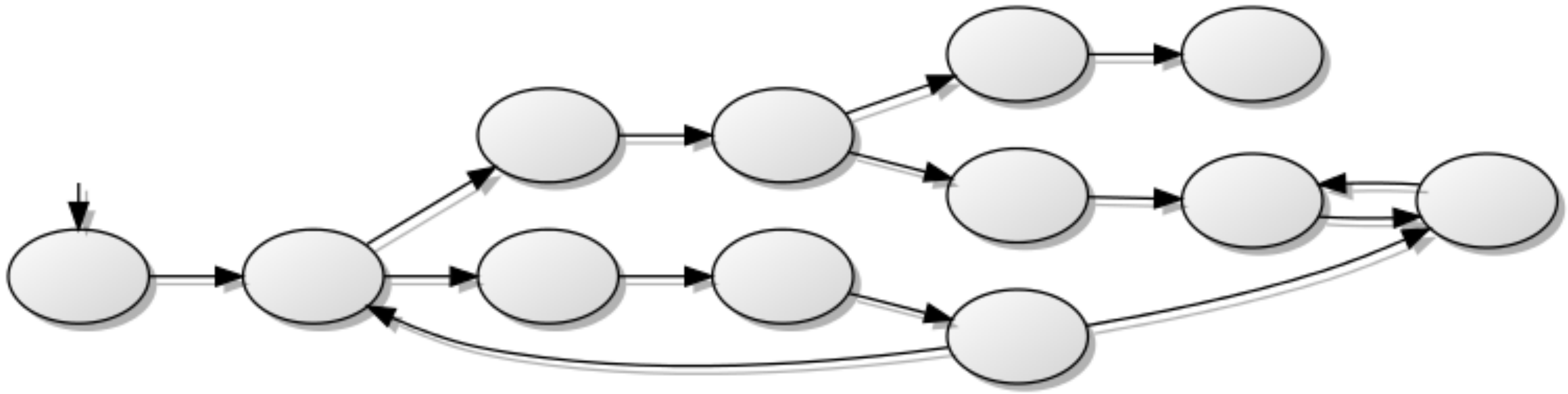
$$\sigma : Addr \to CLO + \mathbf{halt}$$

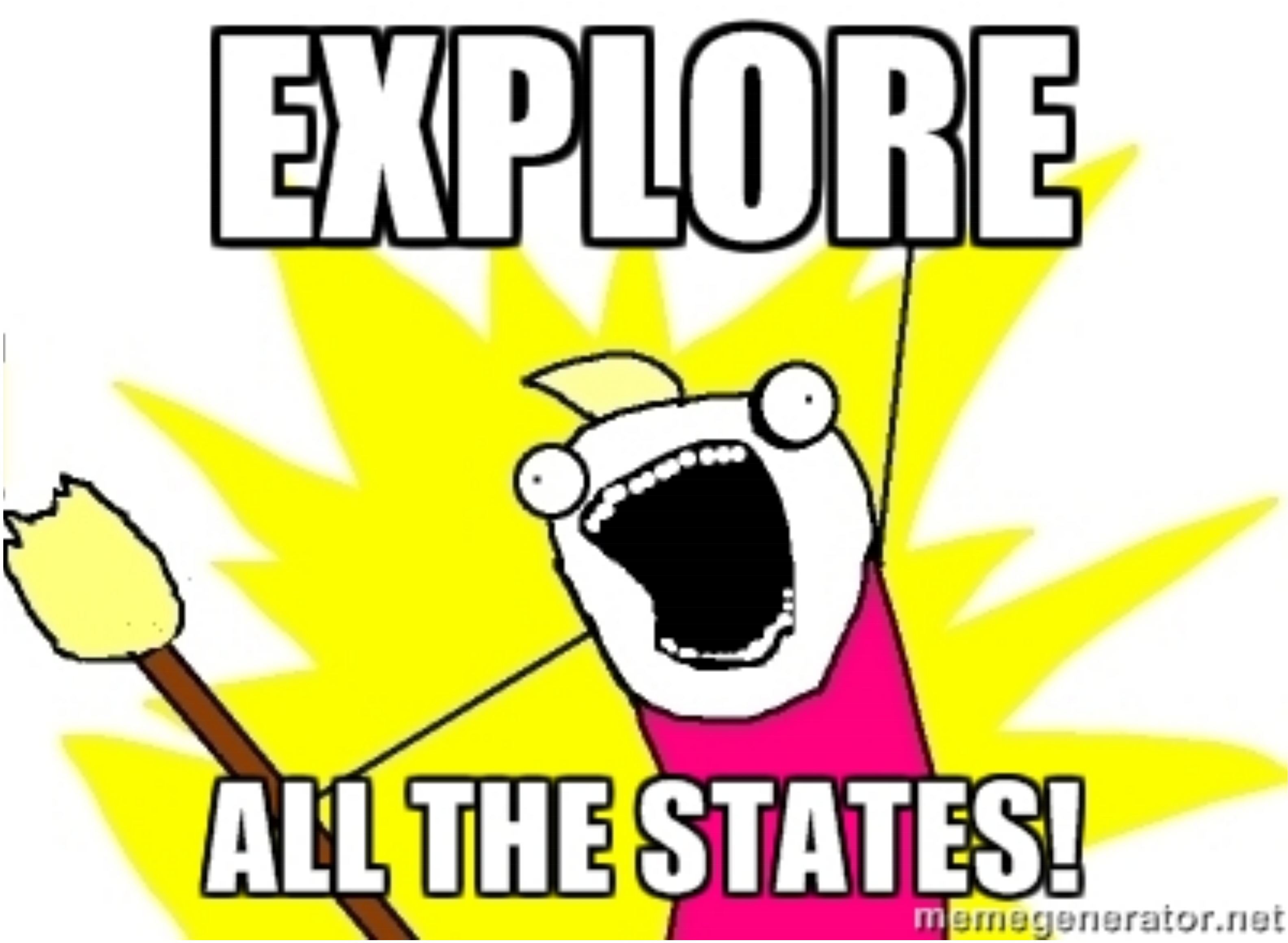
Store

$$\sigma : \widehat{Addr} \rightarrow \mathcal{P}(\widehat{CLO} + \mathbf{halt})$$

CES

CES

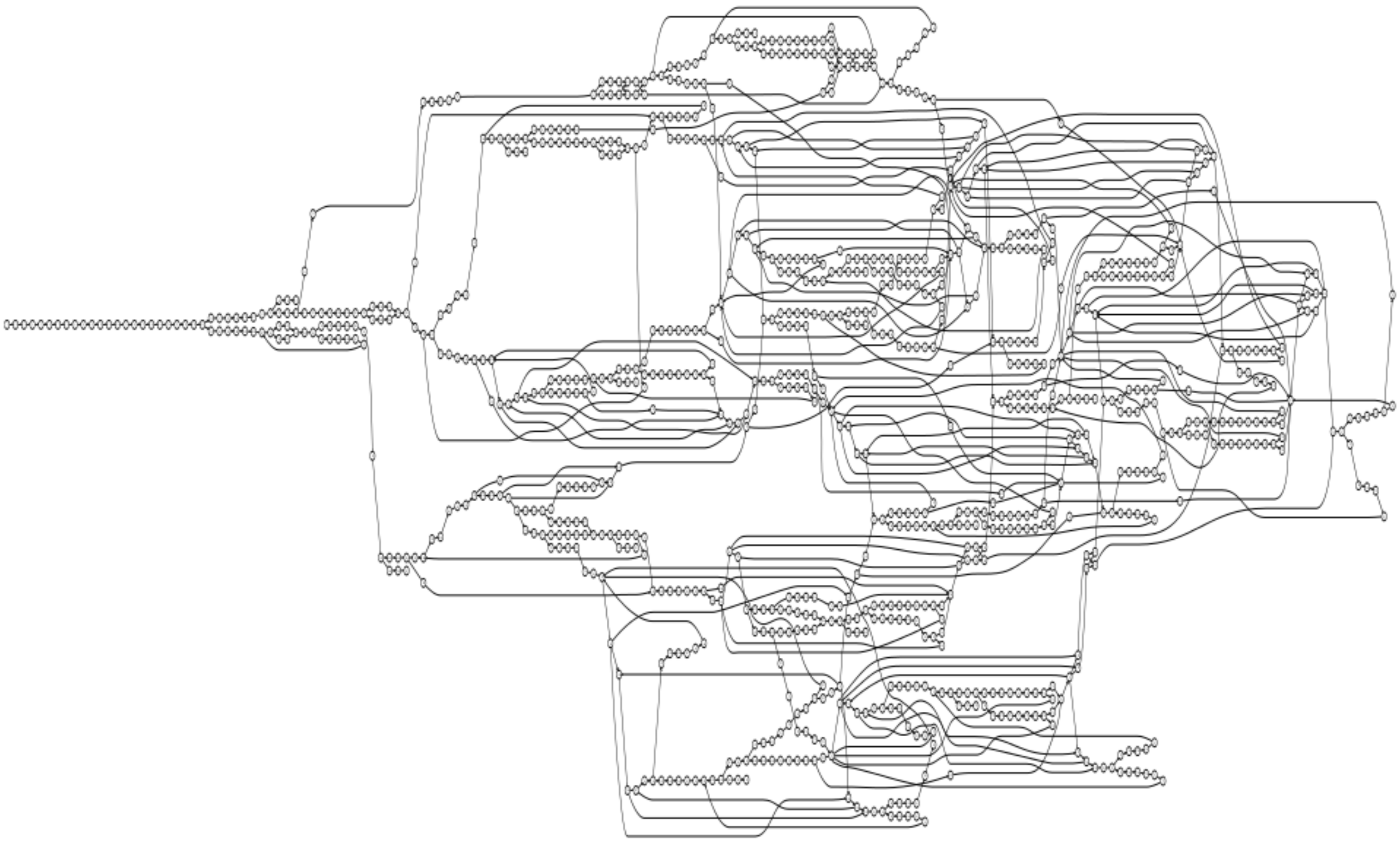


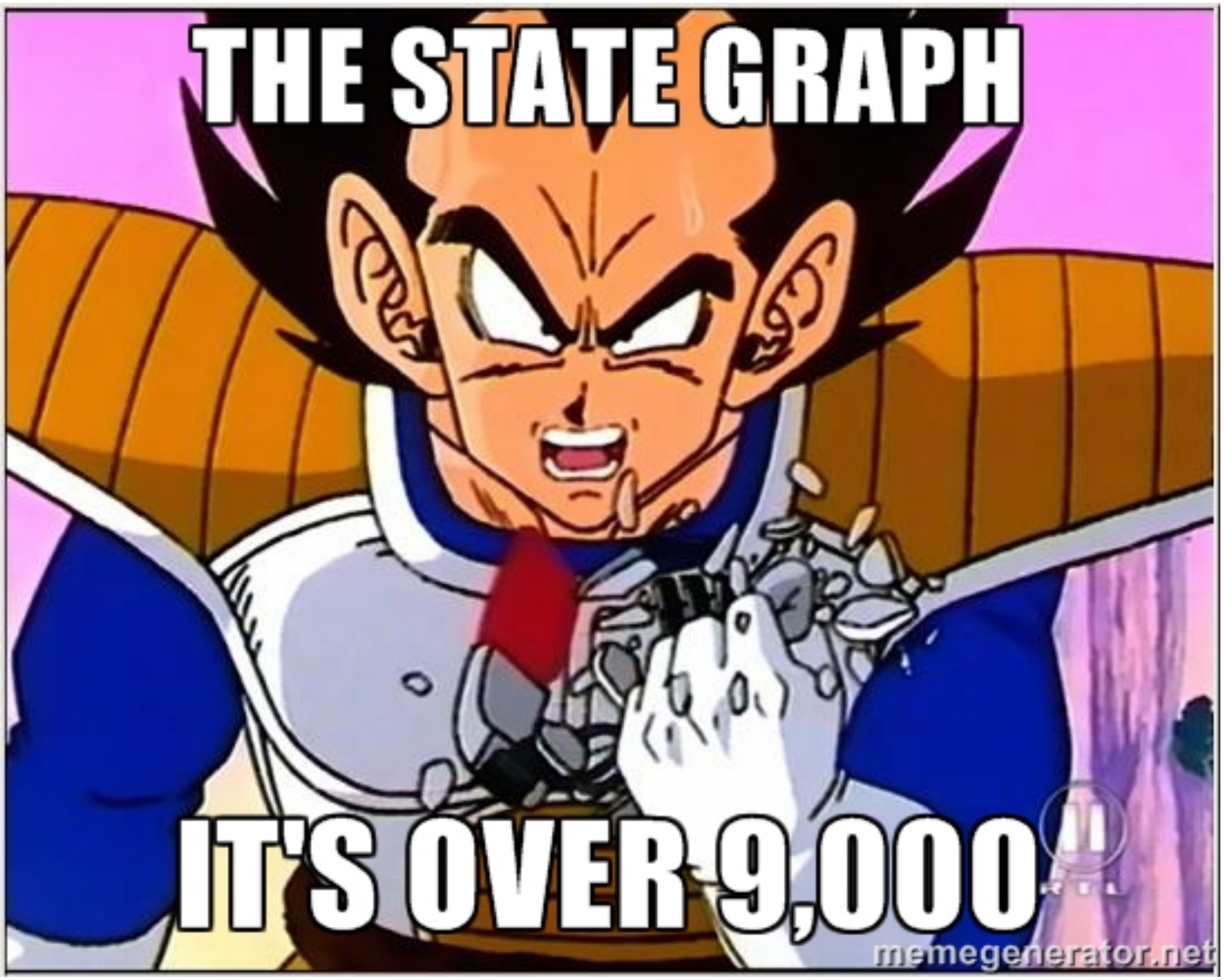

EXPLORE
ALL THE STATES!
memegenerator.net

What is wrong with Control Flow Analysis?

SLOW

```
(letrec ((lp1 (λ (i x)
               (if (= 0 i) x
                   (letrec ((lp2
                             (λ (j f y)
                                (if (= 0 j)
                                    (lp1 (- i 1) y)
                                    (lp2 (- j 1) f
                                         (f y)))))))
                     (lp2 10 (λ (n) (+ n i)) x))))))
  (lp1 10 0))
```




THE STATE GRAPH
IT'S OVER 9,000
memegenerator.net

SLOW

Use Widening

Loses Precision

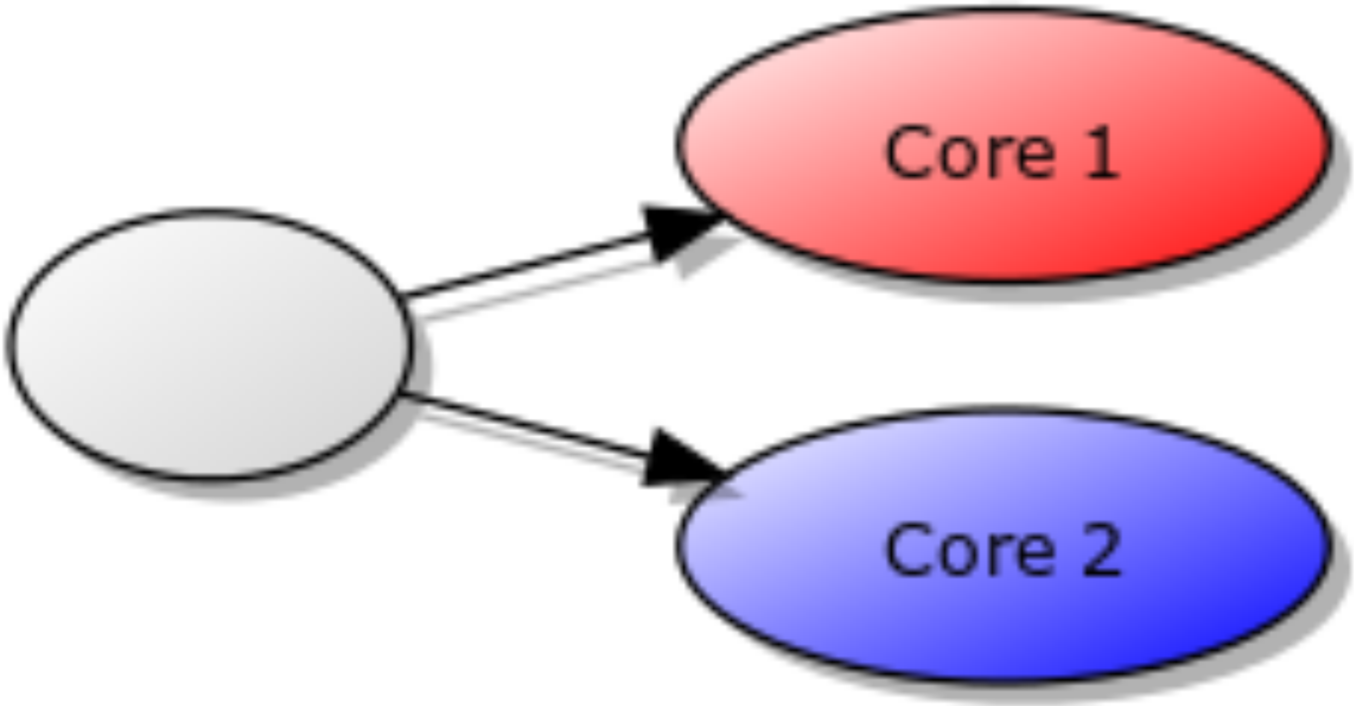
Loses Precision

Solution:

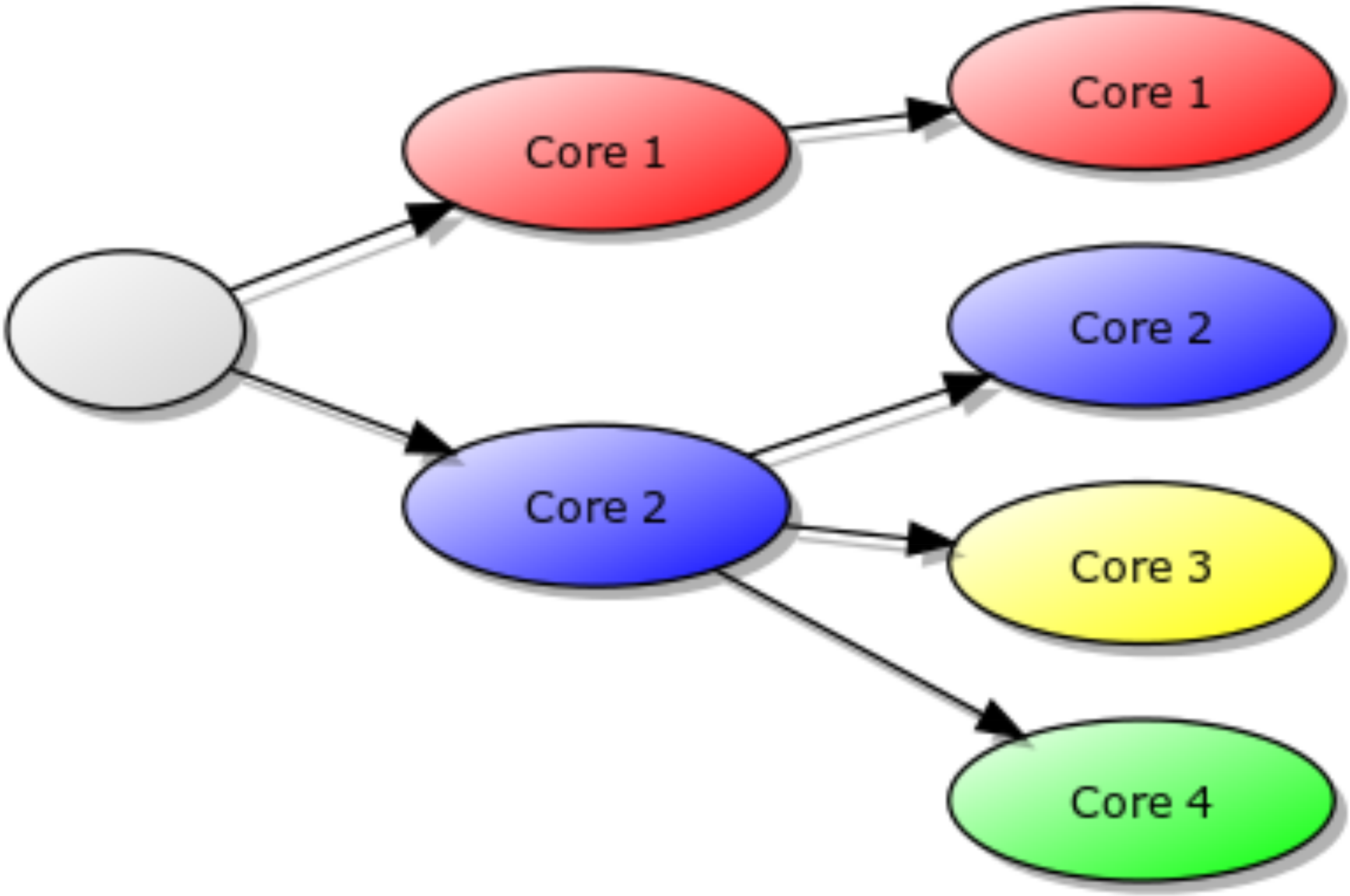
Solution:

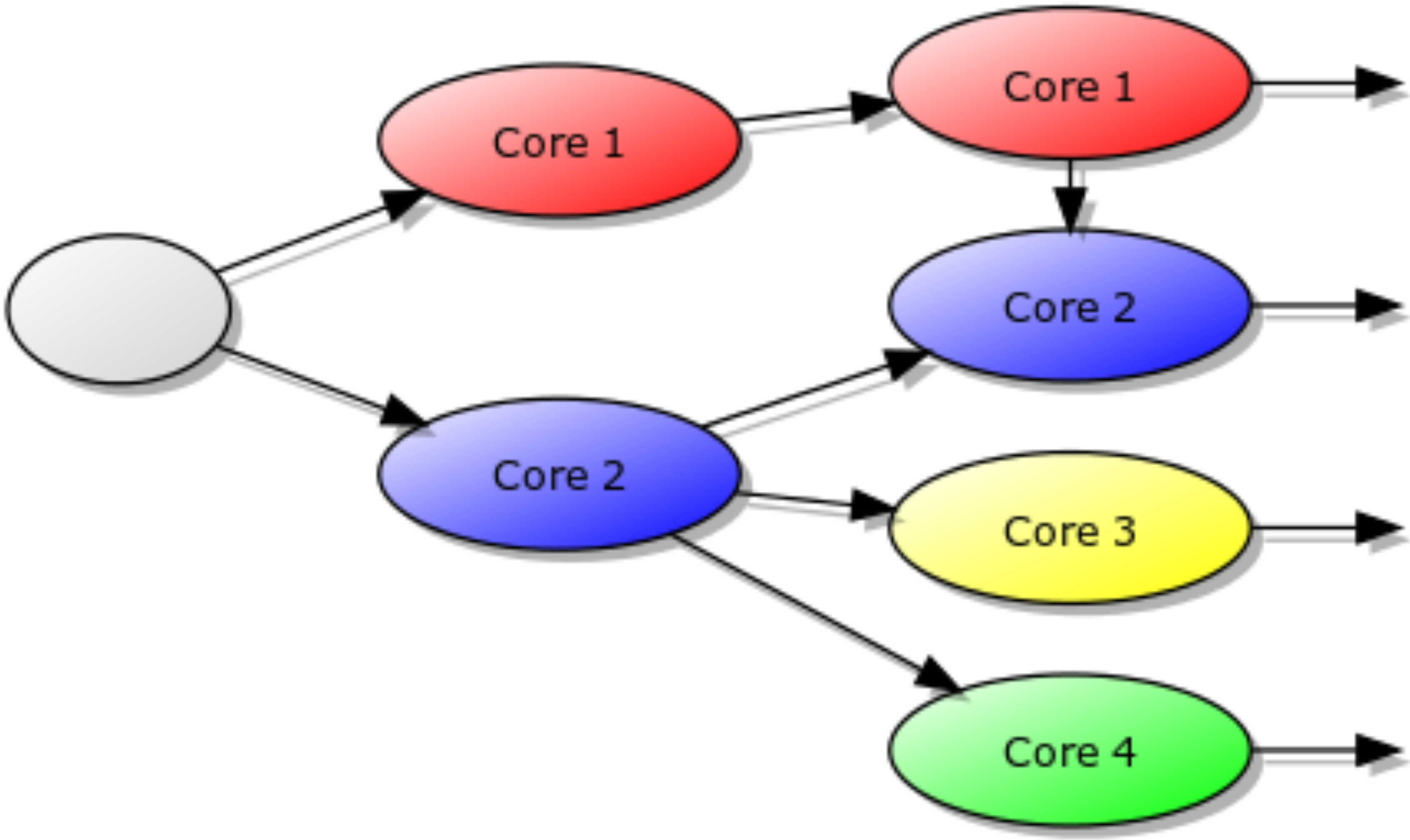
Parallelization

Where?

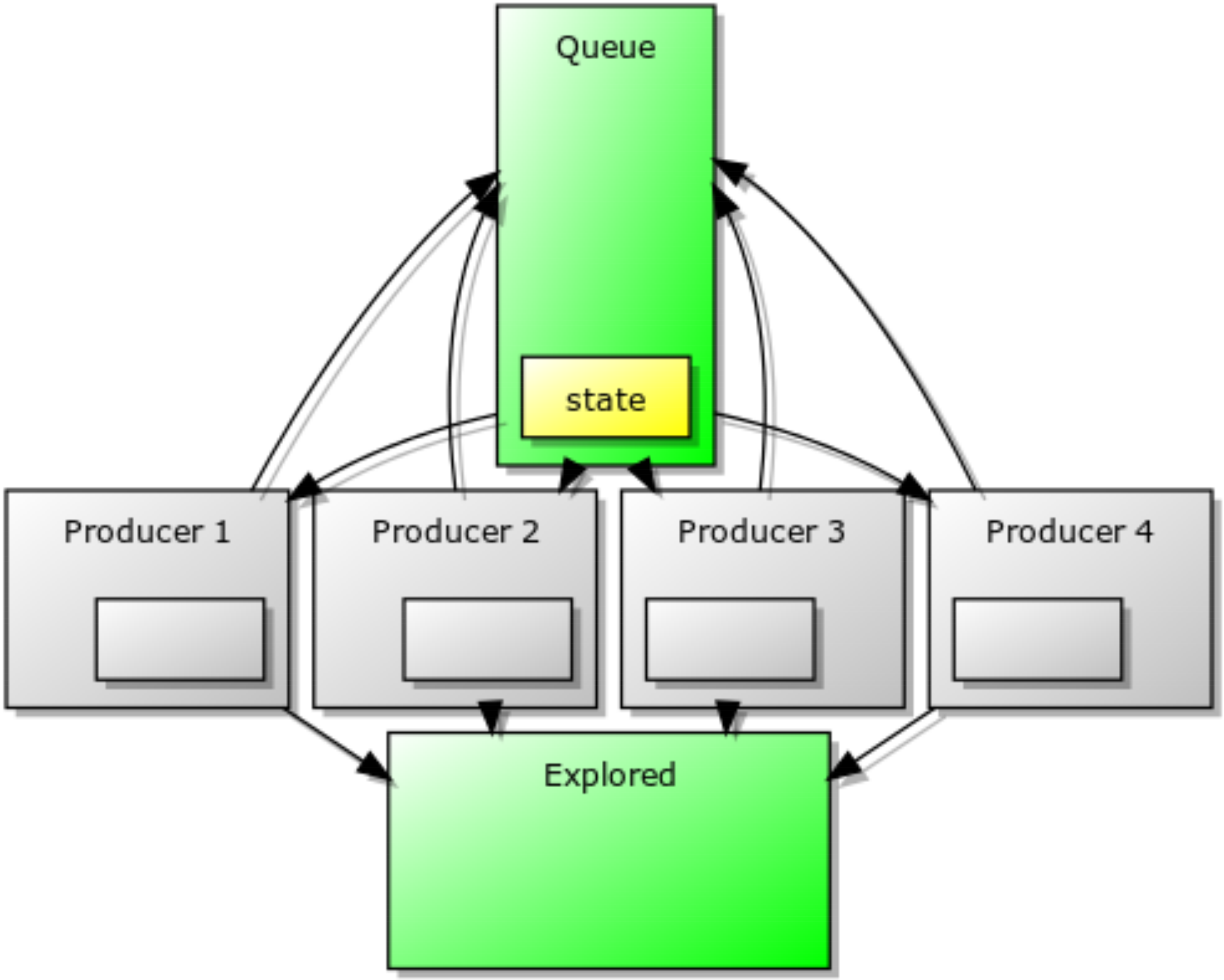


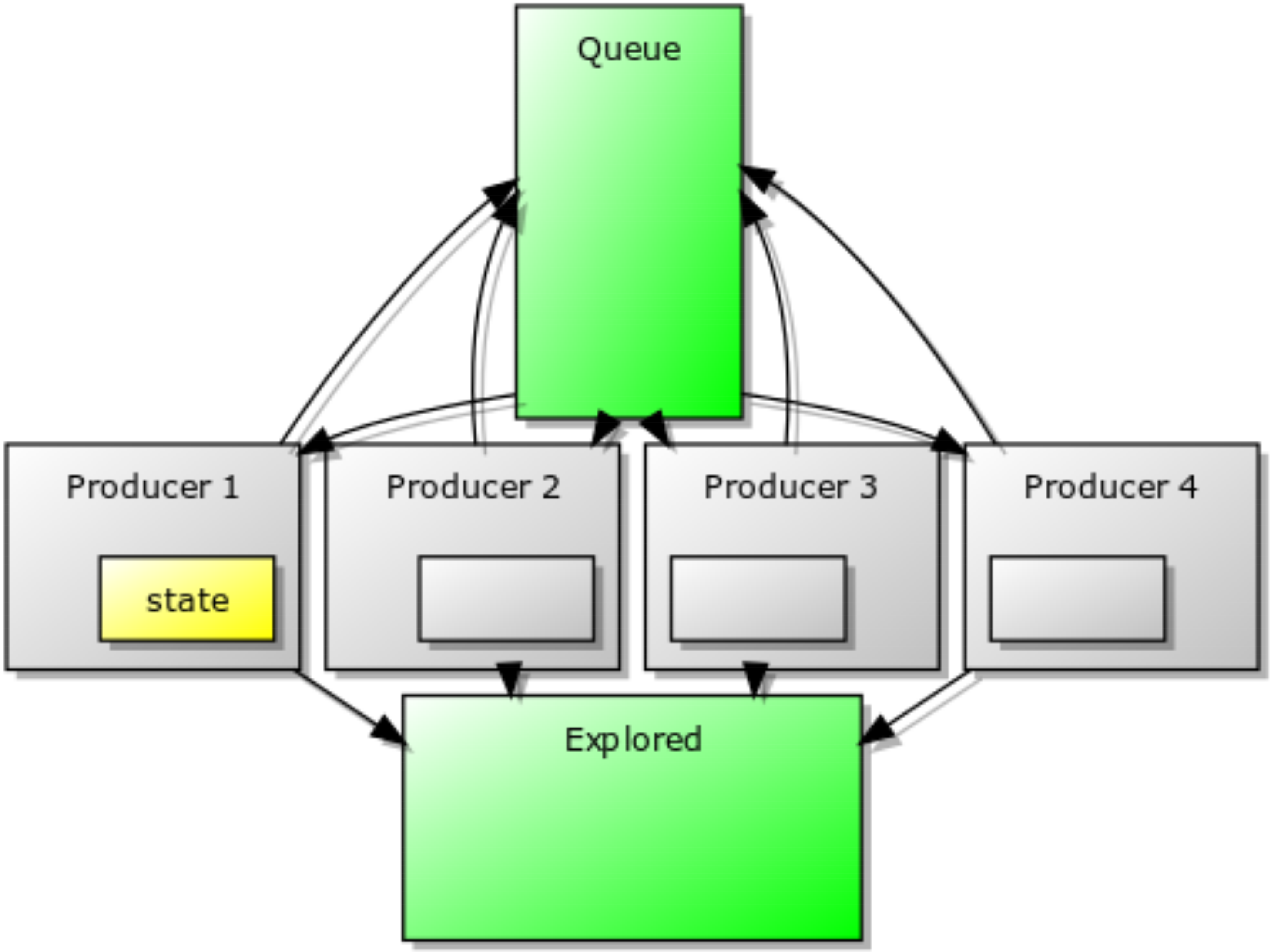
Core 1
Core 2

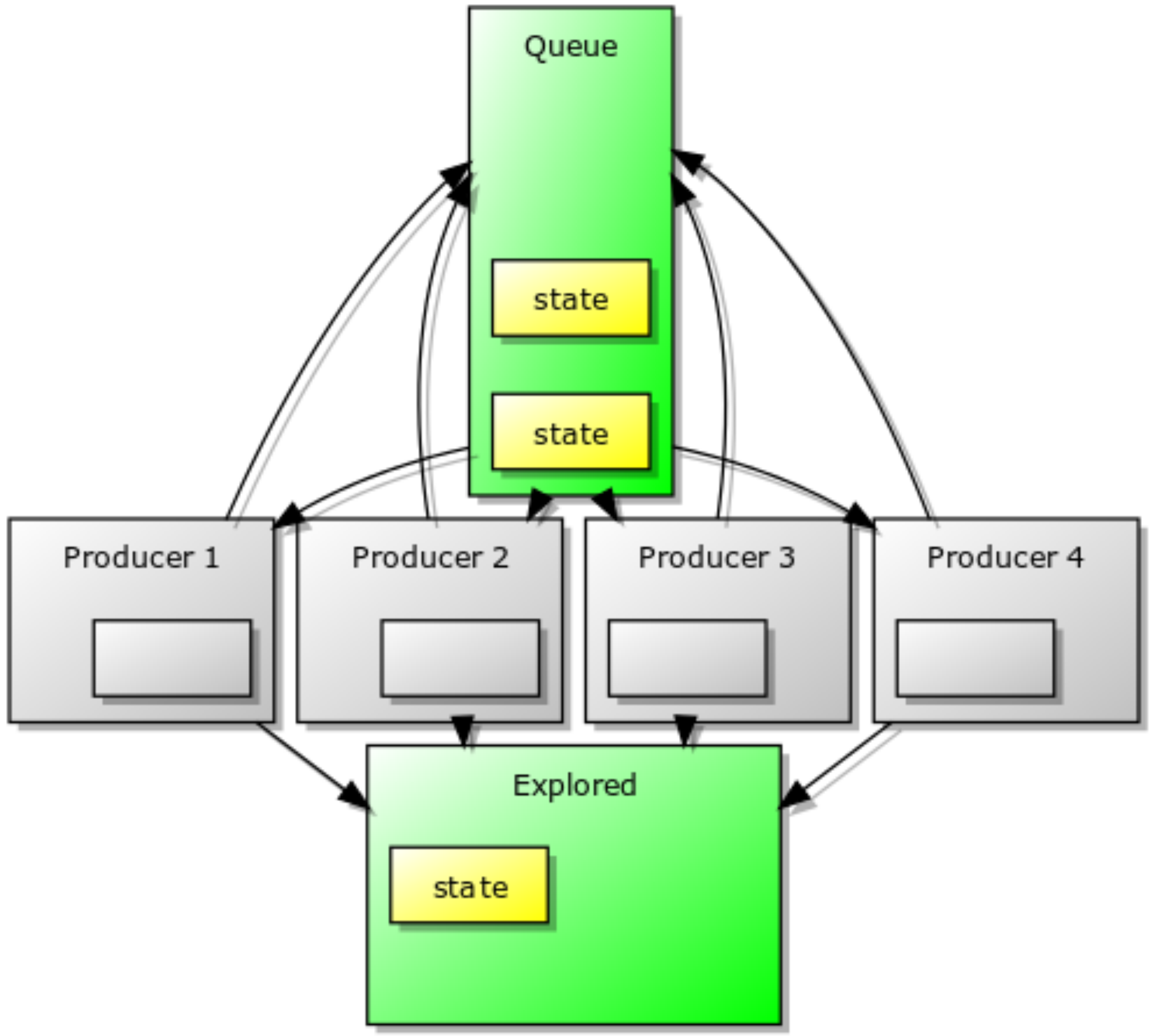

Core 1
Core 1
Core 2
Core 2
Core 3
Core 4

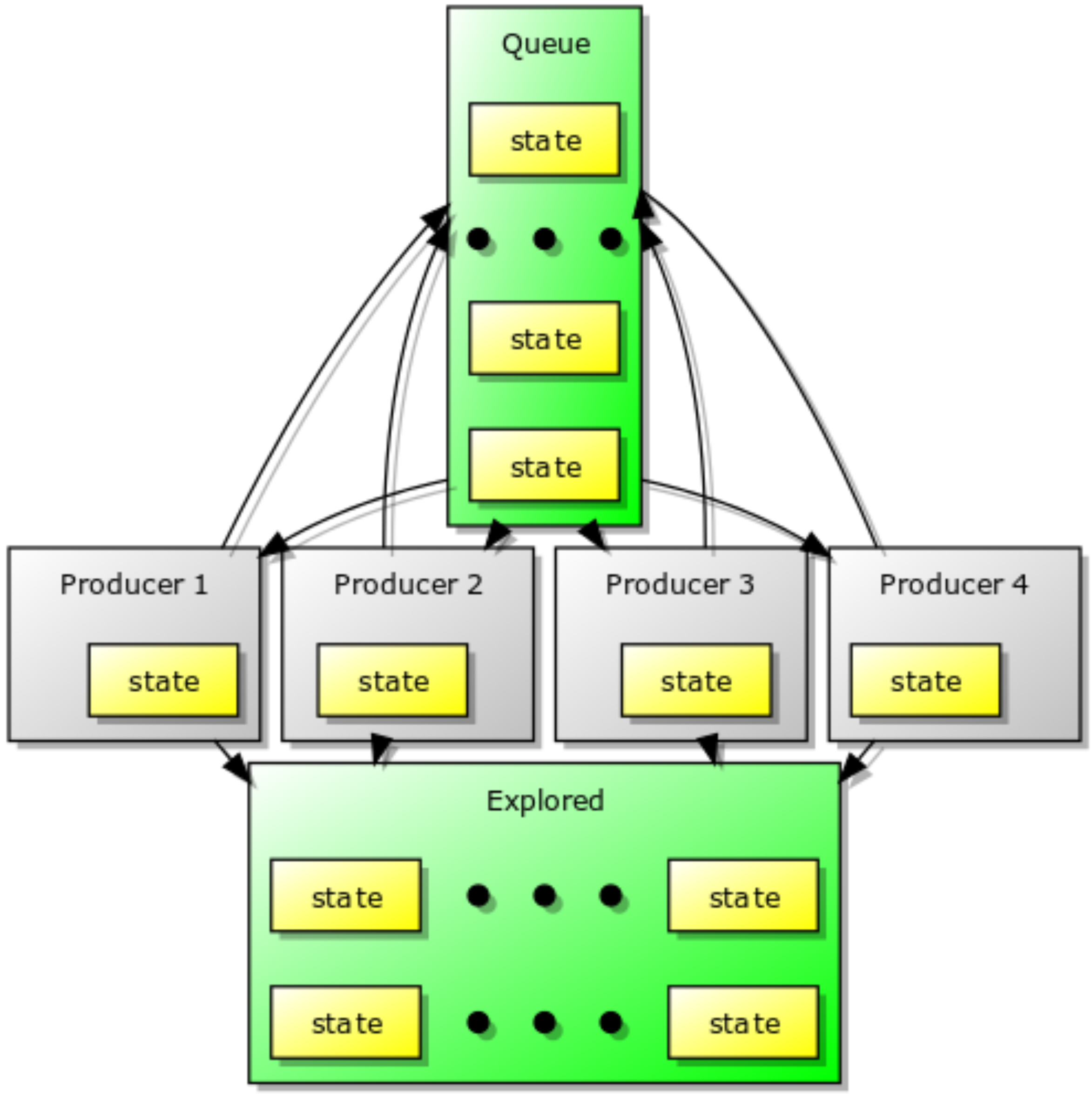

Core 1
Core 1
Core 2
Core 2
Core 3
Core 4

$$\mathcal{I} : \mathsf{Prog} \to \widehat{\Sigma}$$
$$\mathcal{I}(pr) = (pr, [], [])$$


Queue
state
Producer 1
Producer 2
Producer 3
Producer 4
Explored


Queue
Producer 1
state
Producer 2
Producer 3
Producer 4
Explored


Queue
state
state
Producer 1
Producer 2
Producer 3
Producer 4
Explored
state


Queue
state
state
state
Producer 1
state
Producer 2
state
Producer 3
state
Producer 4
state
Explored
state
state
state
state

```
def anaivefix(in: Map[State,Set[State]]):
      Map[State,Set[State]] = {
    var next = in
    for(i <- in.values; j <- i
        if(!next.contains(j))) {
      val step = anaivestep(j)
      next += (j -> step)
    }
    if(in == next) return next
    else              return anaivefix(next)
  }
```

```
def aexplore(in: Map[State,Set[State]]):
    Map[State,Set[State]] = {
  var next = in
  var producers =
    new StateIterator[(State,StateProducer)]
  for(i <- in) getProducers(producers, i._2)
  for(i <- producers) {
    var tmpStep = Set[State]()
    for(j <- i._2.iterator) tmpStep ++= j
    next += (i._1 -> tmpStep)
    getProducers(producers, tmpStep)
  }
  return next
}
```

```
case ApplyState(f, x, s) => {
  val b = for (c <- x) yield aevalState(c)
  for(a <- aevalState(f)
      if (a.e.isInstanceOf[LambExp])
      if (a.e.asInstanceOf[
        LambExp].param.length == b.length))
    yield aapply(a, b, s);
}
```

```
case ApplyState(f, x, s) => {
  val tmpProducers =
    for(c <- x) yield new EvalProducer(c);
  val b = for (c <- tmpProducers) yield {
    var tmpSet = Set[Closure]()
    for(i <- c.iterator) tmpSet ++= i
    tmpSet
  }
  for(a <- aevalState(f))
    yield closureToEval(aapply(a, b, s), s)
}
```

Results

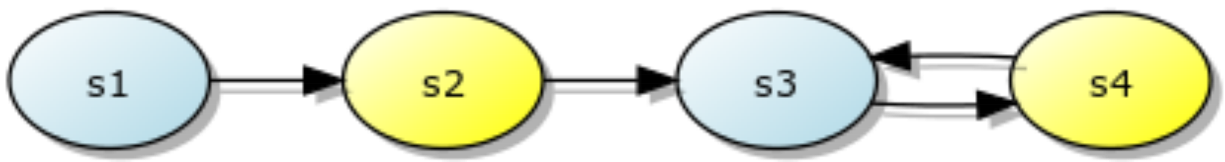
Worst Case:

Worst Case: Omega

```
((λ (x) (x x))
 (λ (x) (x x)))
```

No Speedup

No Speedup



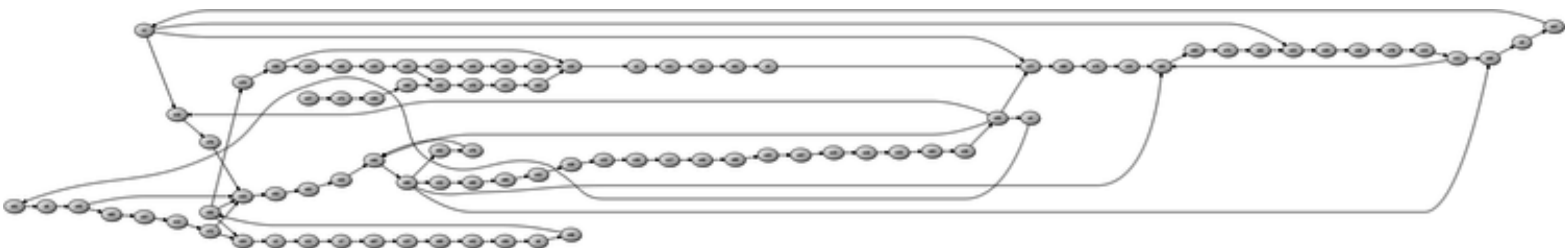
Best Case:

Best Case: Factorial

```
(letrec ((f (λ (n)
             (if (= n 0)
                 1
                 (* n (f (- n 1)))))))
(f 5))
```

Factorial Size	Speedup
10	7.32
15	7.39
20	7.42

Factorial Size	Speedup
10	7.32
15	7.39
20	7.42



Average Case:

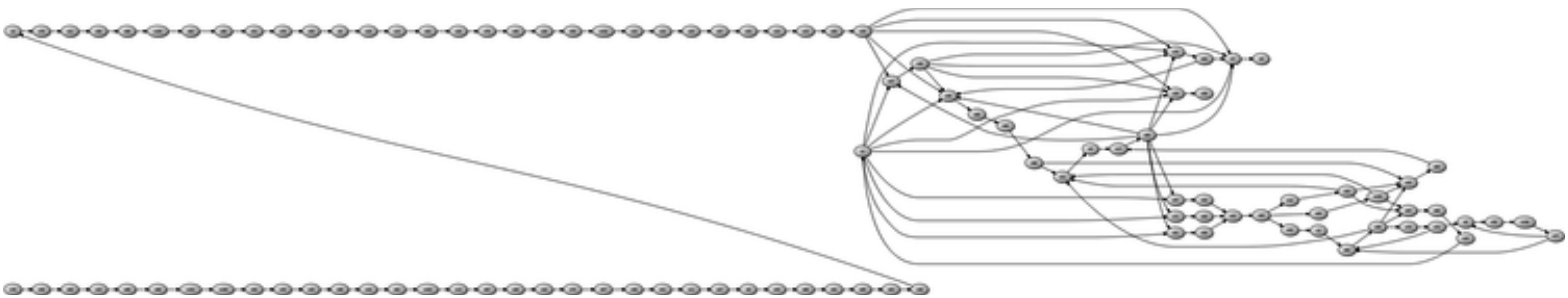
Average Case:

Collatz Conjecture

```
(letrec ((even? (λ (n)
                  (if (= n 0)
                    #t
                    (if (= n 1)
                      #f
                      (even? (- n 2)))))))
  (letrec ((div2* (λ (n s)
                    (if (= (* 2 n) s)
                      n
                      (if (= (+ (* 2 n) 1) s)
                        n
                        (div2* (- n 1) s))))))
    (letrec ((div2 (λ (n)
                     (div2* n n))))
      (letrec ((hailstone* (λ (n count)
                             (if (= n 1)
                               count
                               (if (even? n)
                                 (hailstone*
                                   (div2 n) (+ count 1))
                                 (hailstone*
                                   (+ (* 3 n) 1)
                                   (+ count 1)))))))
        (letrec ((hailstone (λ (n)
                              (hailstone* n 0))))
          (hailstone 5))))))
```

Hailstone Size	Speedup
5	2.3

Hailstone Size	Speedup
5	2.3



Implementation

Implementation

https://github.com/LeifAndersen/CPSLambdaCalc


I DON'T ALWAYS ANALYZE MY CODE
BUT WHEN I DO, I DO IT IN PARALLEL
memegenerator.net

Questions?

Where?

- Explore

Where?

- Explore
- Function Application

$$explore : \Sigma \to \mathcal{P}(\Sigma)$$

$$explore(\varsigma) = \{\varsigma' \mid \varsigma \to^* \varsigma'\}$$

$$\mathcal{A} : \mathsf{ENV} \times \widehat{ENV} \times \widehat{Store}$$

$$[((\lambda\ (v_1 \ldots v_n)\ ce)], \rho') = \mathcal{A}(f, \rho, \sigma)$$