Petri Net-Flavored Places

An Advanced Transition System for Distributed Computing in Racket

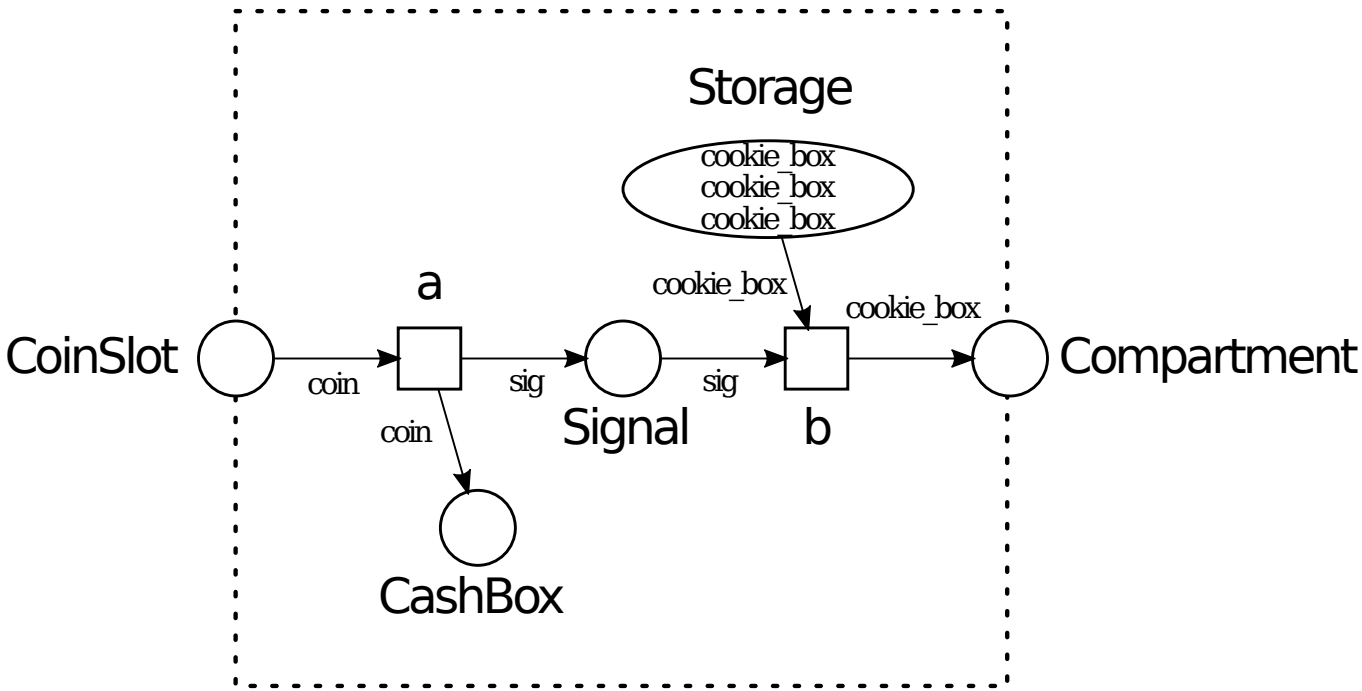
Jörgen Brandt

2018-09-29

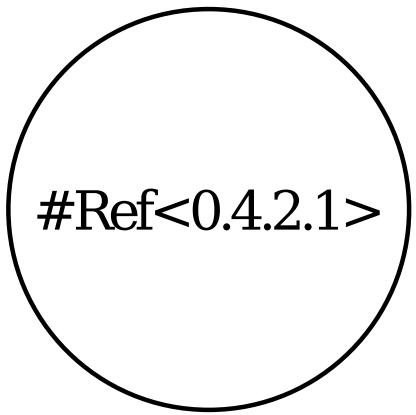
Software Engineering

Petri Nets

- Visually understandable
- Defined semantics (properties, invariants, correctness)
- Complete (can be executed, tested)

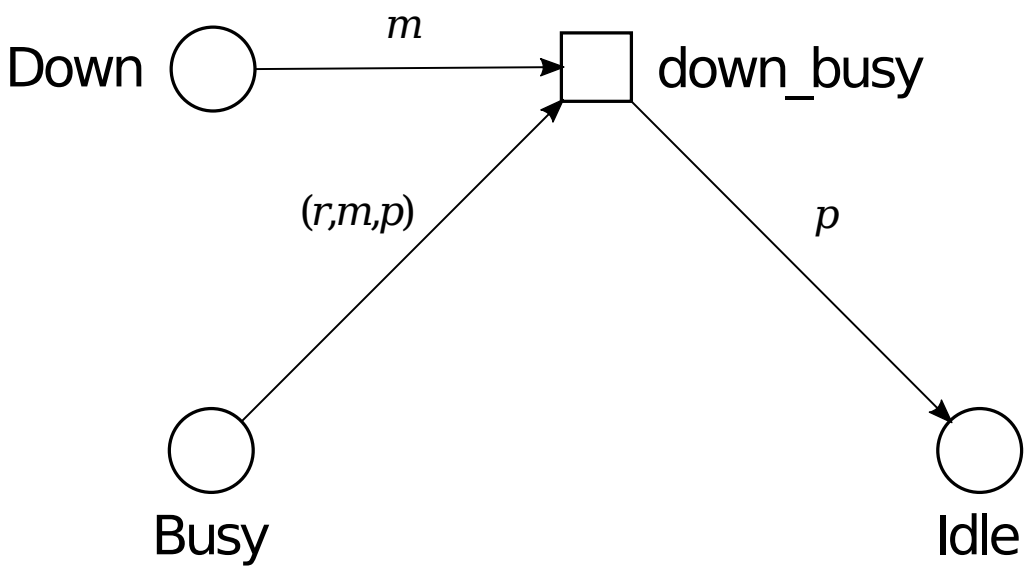


Petri Net Syntax



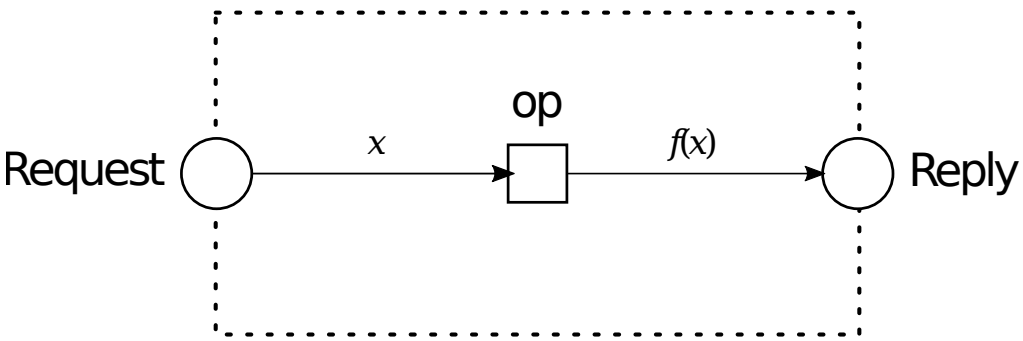
- Passive component: Place

Petri Net Syntax



- Passive component: Place
- Active component: Transition

Petri Net Syntax

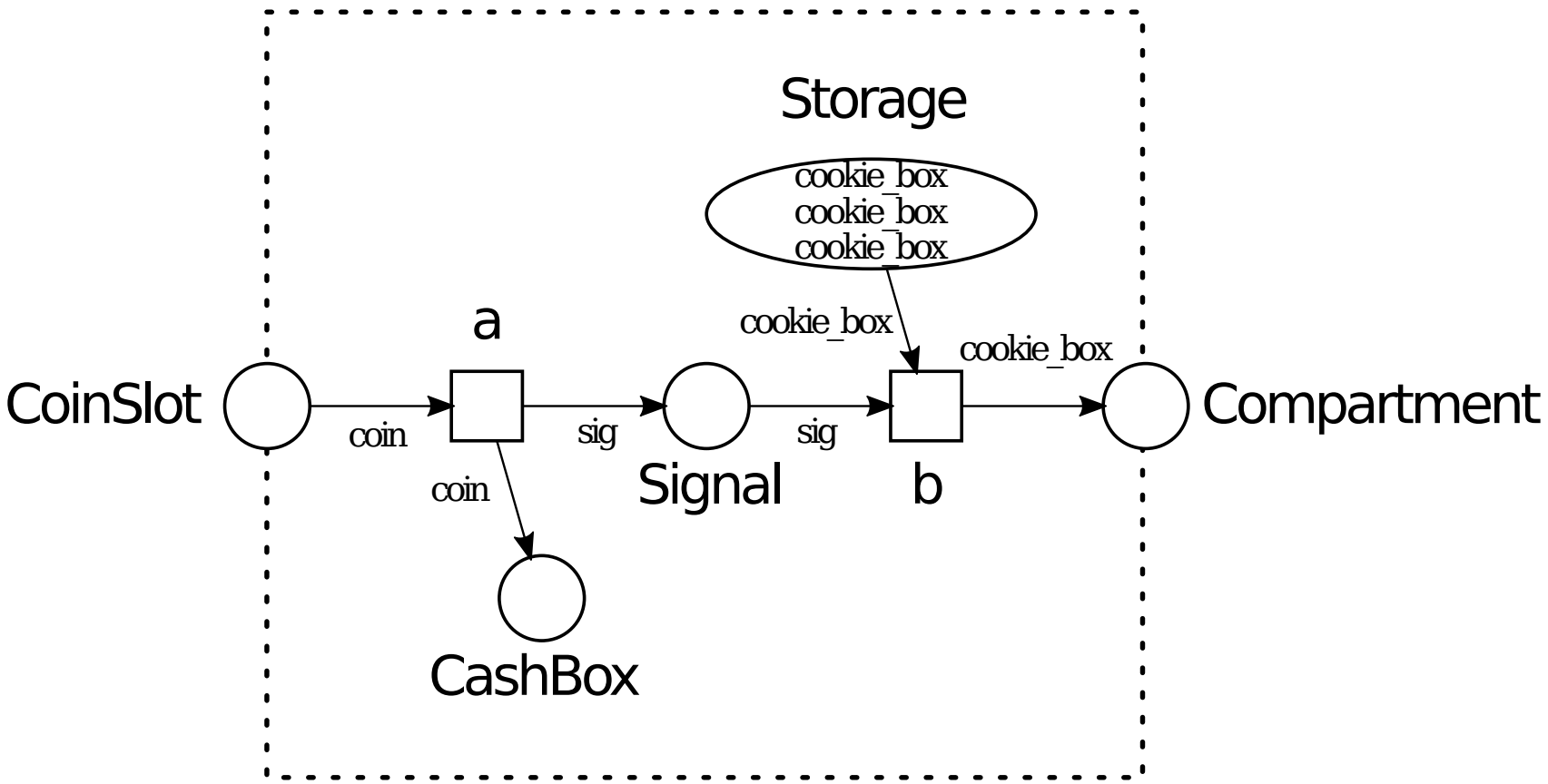


- Passive component: Place
- Active component: Transition
- Environment

"High-level interface nets"

Example

Implementation: Cookie Vending Machine



`https://github.com/joergen7/pnet`

Implementation: Cookie Vending Machine

```
(define place-set : (Setof Symbol)
  (set 'coin-slot 'cash-box 'signal 'storage 'compartment))
```

Implementation: Cookie Vending Machine

```
(define place-set : (Setof Symbol)
  (set 'coin-slot 'cash-box 'signal 'storage 'compartment))

(define preset-hash : (HashTable Symbol (Listof Symbol))
  (hash 'a '(coin-slot)
        'b '(signal storage)))
```

Implementation: Cookie Vending Machine

```
(define place-set : (Setof Symbol)
  (set 'coin-slot 'cash-box 'signal 'storage 'compartment))

(define preset-hash : (HashTable Symbol (Listof Symbol))
  (hash 'a '(coin-slot)
        'b '(signal storage)))

(: init-marking (Symbol Any -> (Listof Any)))
(define (init-marking place usr-info)
  (match place
    ['storage '(cookie-box cookie-box cookie-box)]
    [_        '()]))
```

Implementation: Cookie Vending Machine

```
(define place-set : (Setof Symbol)
  (set 'coin-slot 'cash-box 'signal 'storage 'compartment))

(define preset-hash : (HashTable Symbol (Listof Symbol))
  (hash 'a '(coin-slot)
        'b '(signal storage)))

(: init-marking (Symbol Any -> (Listof Any)))
(define (init-marking place usr-info)
  (match place
    ['storage '(cookie-box cookie-box cookie-box)]
    [_        '()]))

(: enabled? (Symbol Mode Any -> Boolean))
(define (enabled? trsn mode usr-info)
  #t)
```

Implementation: Cookie Vending Machine

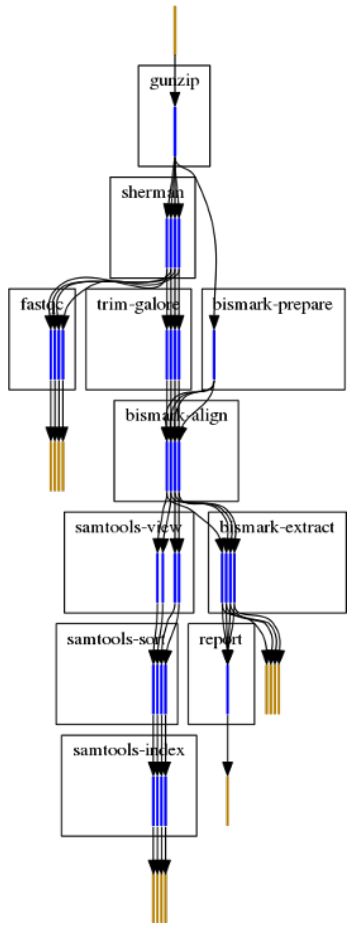
```
(define place-set : (Setof Symbol)
  (set 'coin-slot 'cash-box 'signal 'storage 'compartment))

(define preset-hash : (HashTable Symbol (Listof Symbol))
  (hash 'a '(coin-slot)
        'b '(signal storage)))

(: init-marking (Symbol Any -> (Listof Any)))
(define (init-marking place usr-info)
  (match place
    ['storage '(cookie-box cookie-box cookie-box)]
    [_        '()]))

(: enabled? (Symbol Mode Any -> Boolean))
(define (enabled? trsn mode usr-info)
  #t)

(: fire (Symbol Mode Any -> Mode))
(define (fire trsn mode usr-info)
  (match trsn
    ['a (hash 'signal '(sig) 'cash-box '(coin))]
    ['b (hash 'compartment '(cookie-box))]))
```

Possible Application: Cuneiform

Cuneiform: Motivation

- Cuneiform is ...
  - Functional programming language
  - Distributed language
  - Integration of other languages
- Open:
  - command line tools
  - R scripts
  - Python libraries
  - ...
- General:
  - Universal model of computation

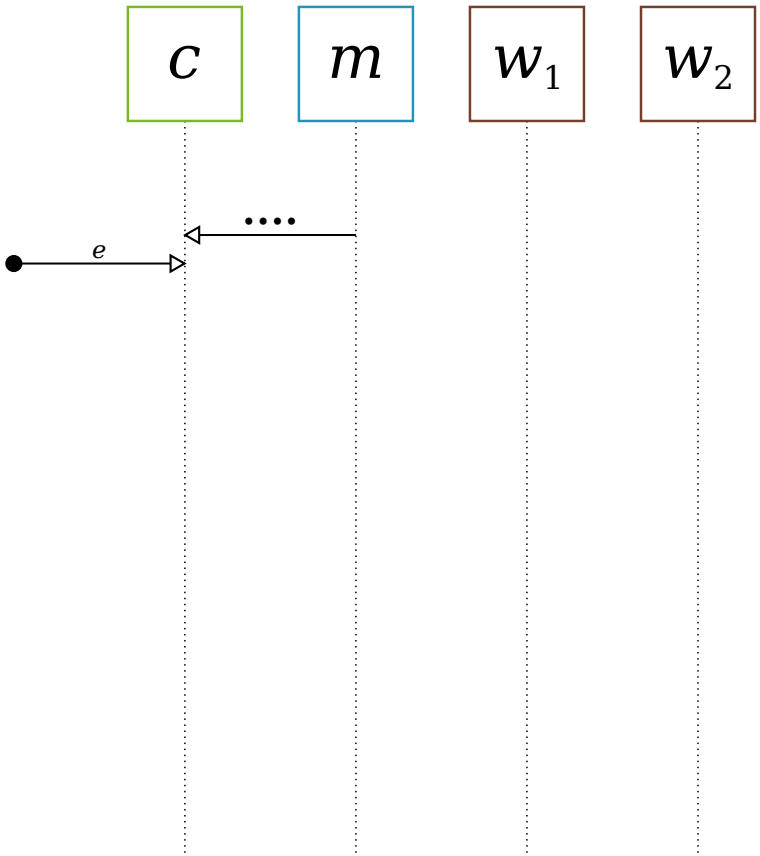
`https://cuneiform-lang.org`



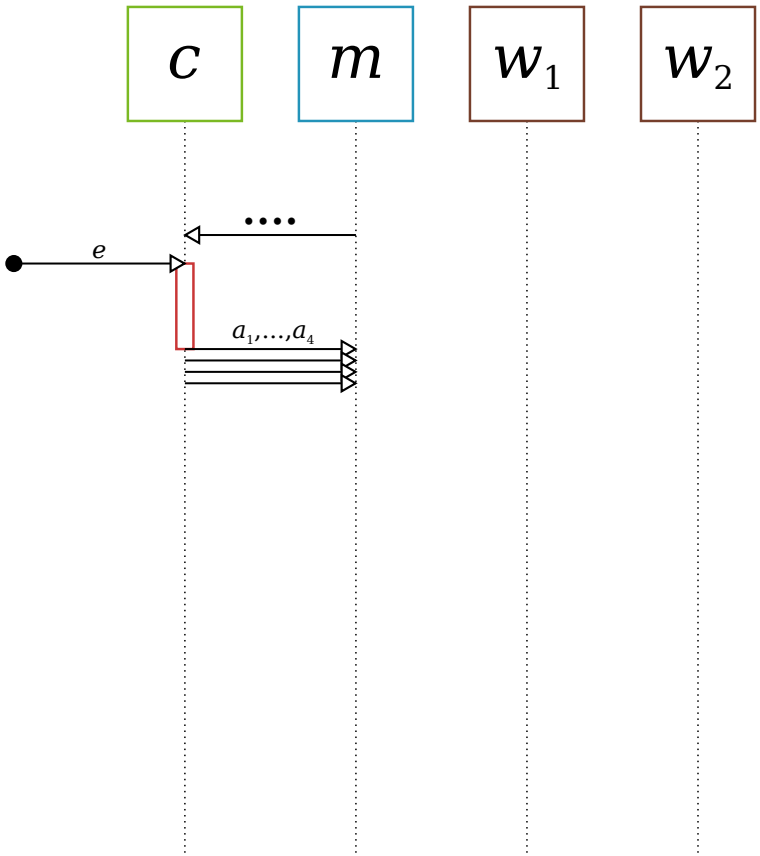
Cuneiform Code Example: Iteration

```
def f( txt : File ) -> <y : File> in Perl *{
  ...
}*

let xs : [File] = ['a.txt', 'b.txt' : File];

for x <- xs do
  f( txt = x )|y
end;
```

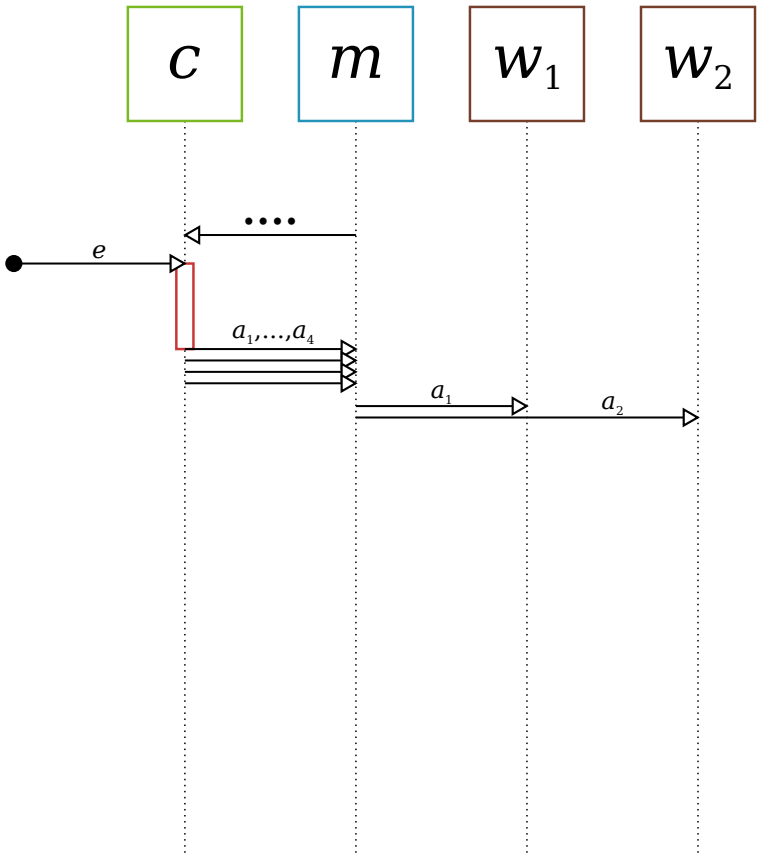
Distrubuted Execution Environment: Sequence Diagram



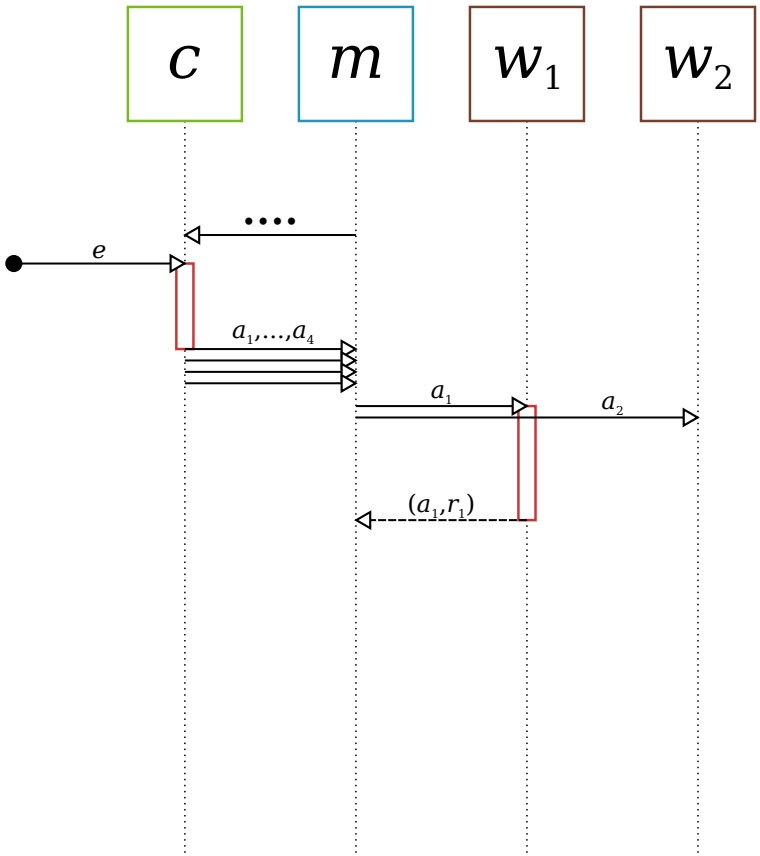
Distributed Execution Environment: Sequence Diagram



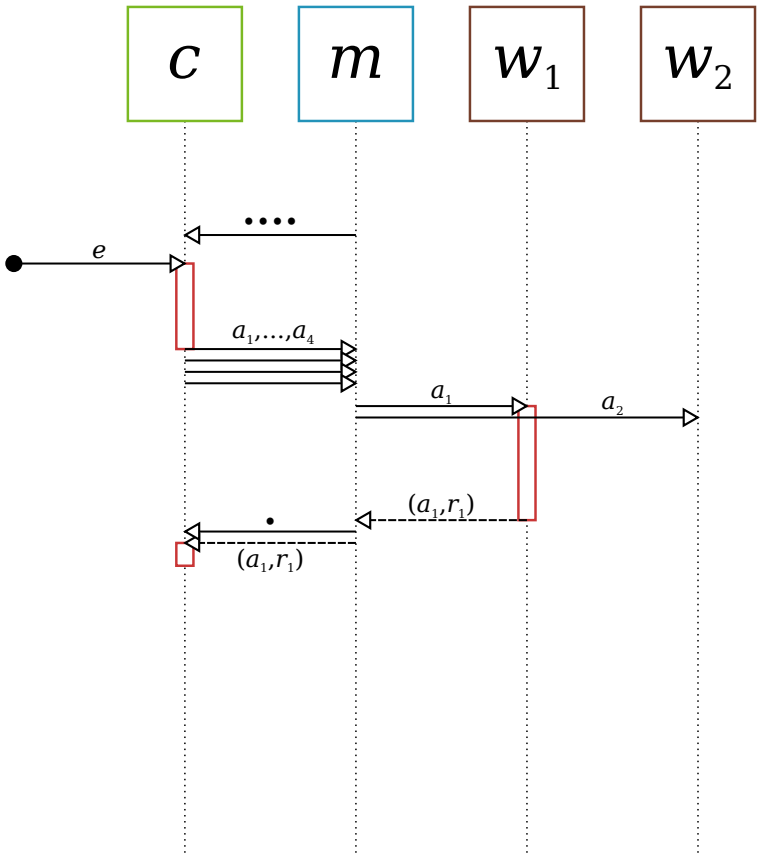
Distributed Execution Environment: Sequence Diagram



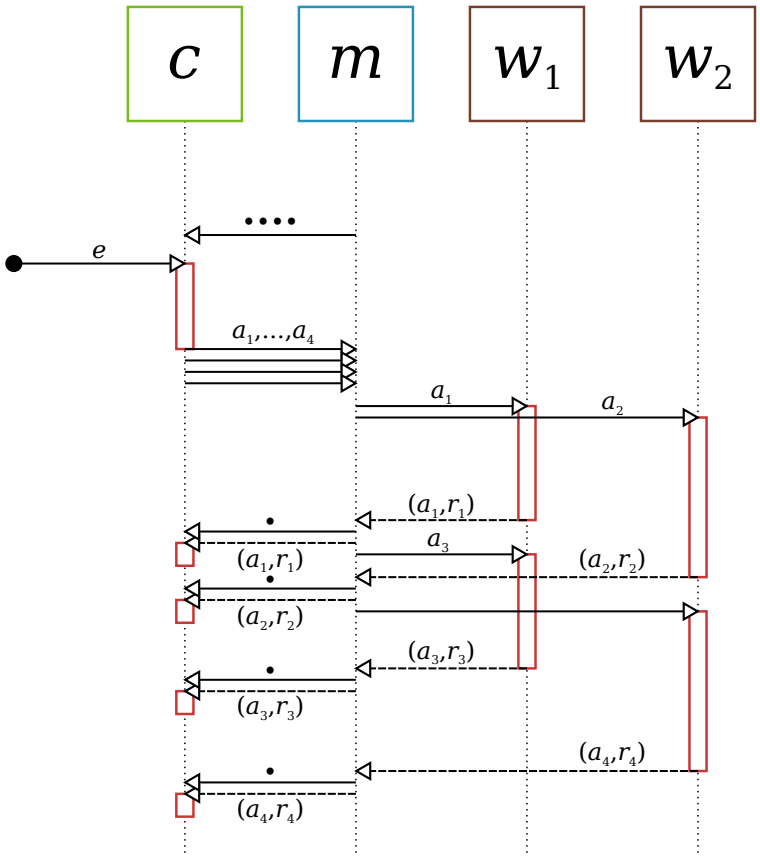
Distributed Execution Environment: Sequence Diagram



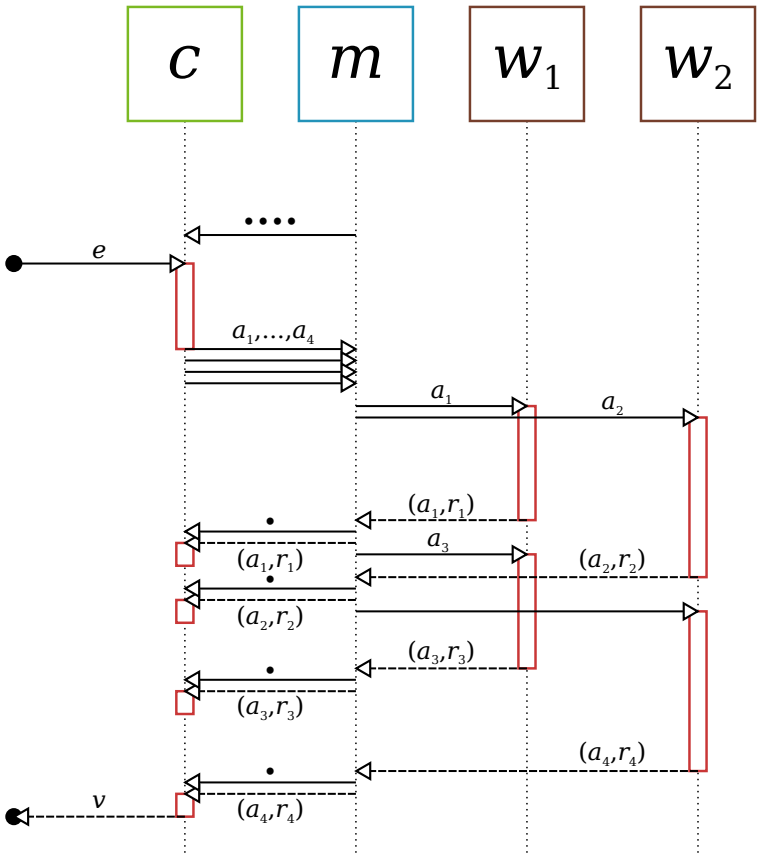
Distributed Execution Environment: Sequence Diagram



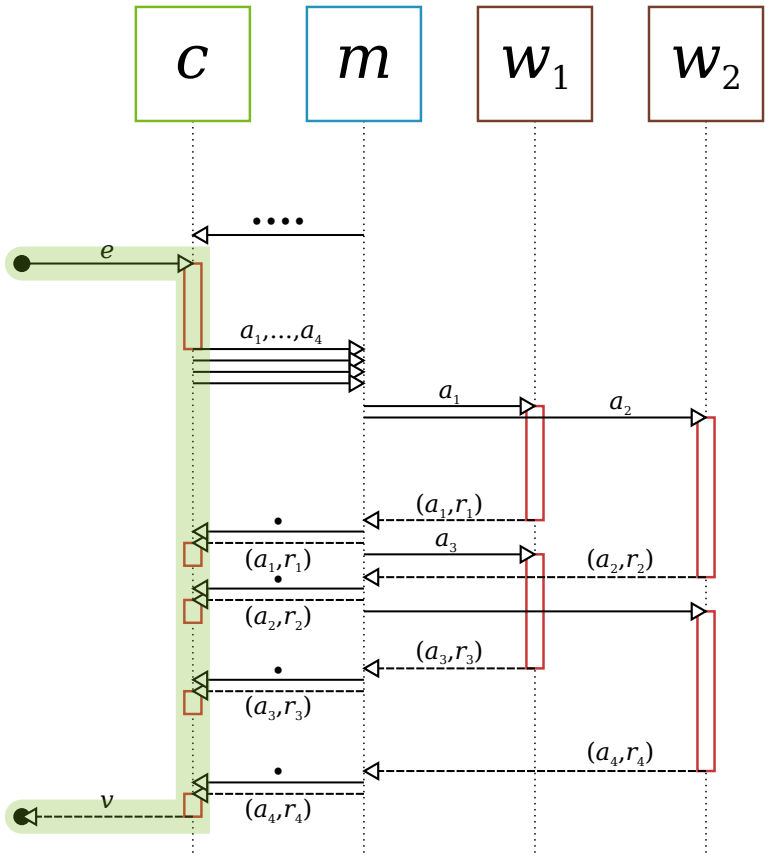
Distributed Execution Environment: Sequence Diagram



Distributed Execution Environment: Sequence Diagram



Distributed Execution Environment: Sequence Diagram



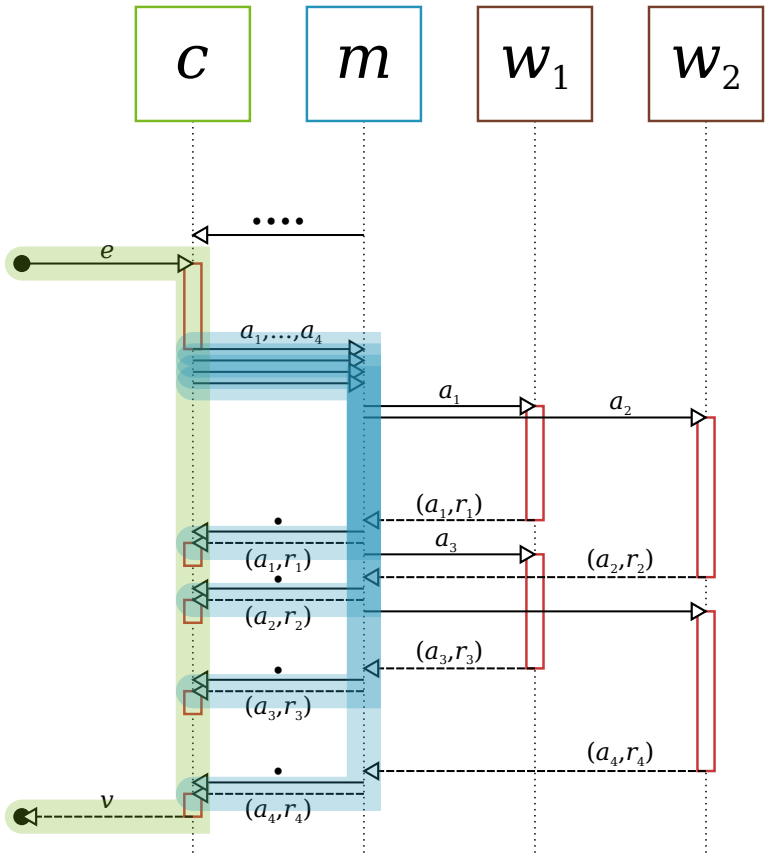
Three interfaces

Between . . .

- User and client

  expressions, values

Distributed Execution Environment: Sequence Diagram



Three interfaces

Between . . .

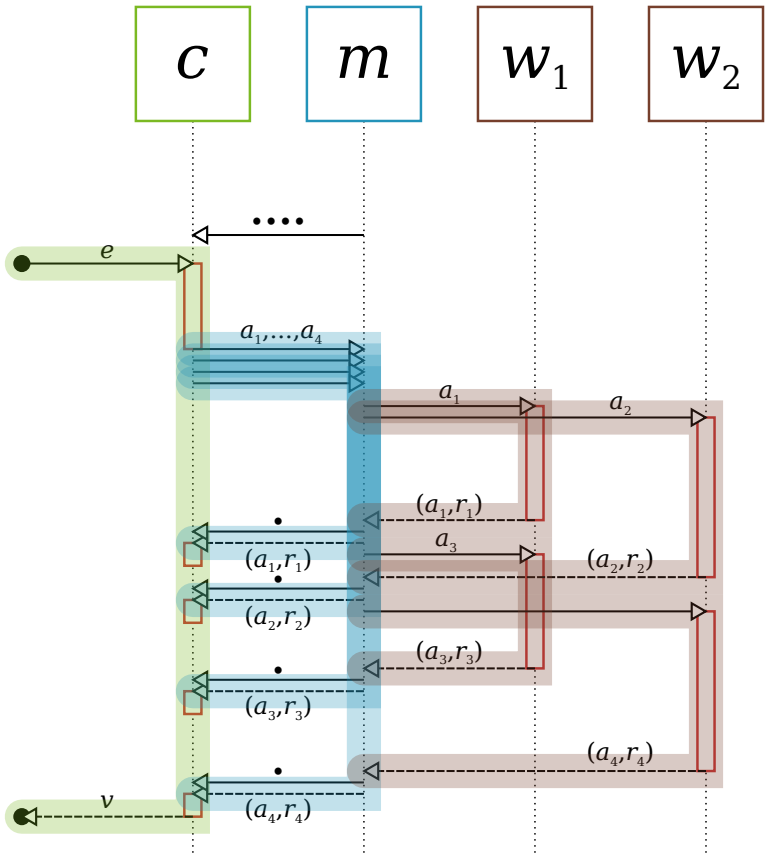
- User and client

  expressions, values

- Client and master

  demand, applications, results

Distributed Execution Environment: Sequence Diagram



Three interfaces

Between . . .

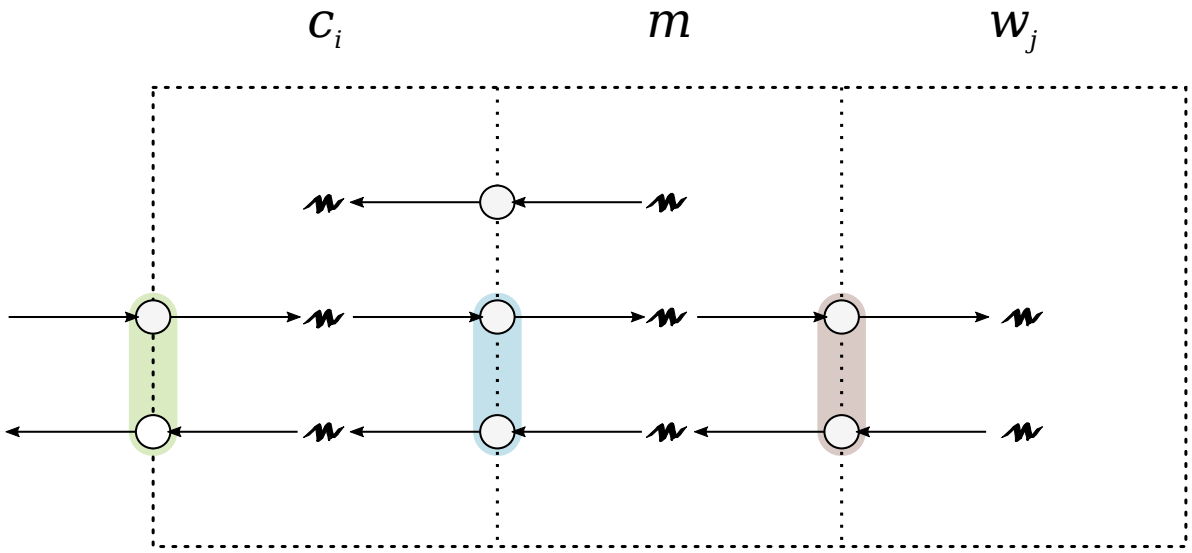
- User and client

  expressions, values
- Client and master

  demand, applications, results
- Master and worker

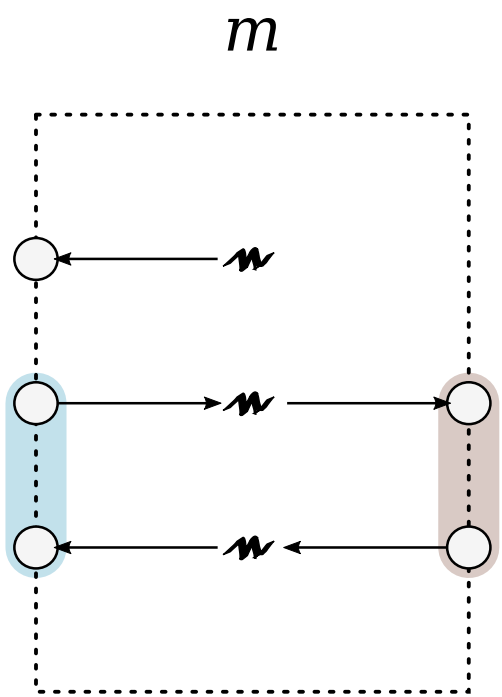
  applications, results

Distributed Execution Environment: Petri Net Model



- The sequence diagram suggests a coarse structure
- Composing nets in a distributed system
- m independent clients and n independent workers

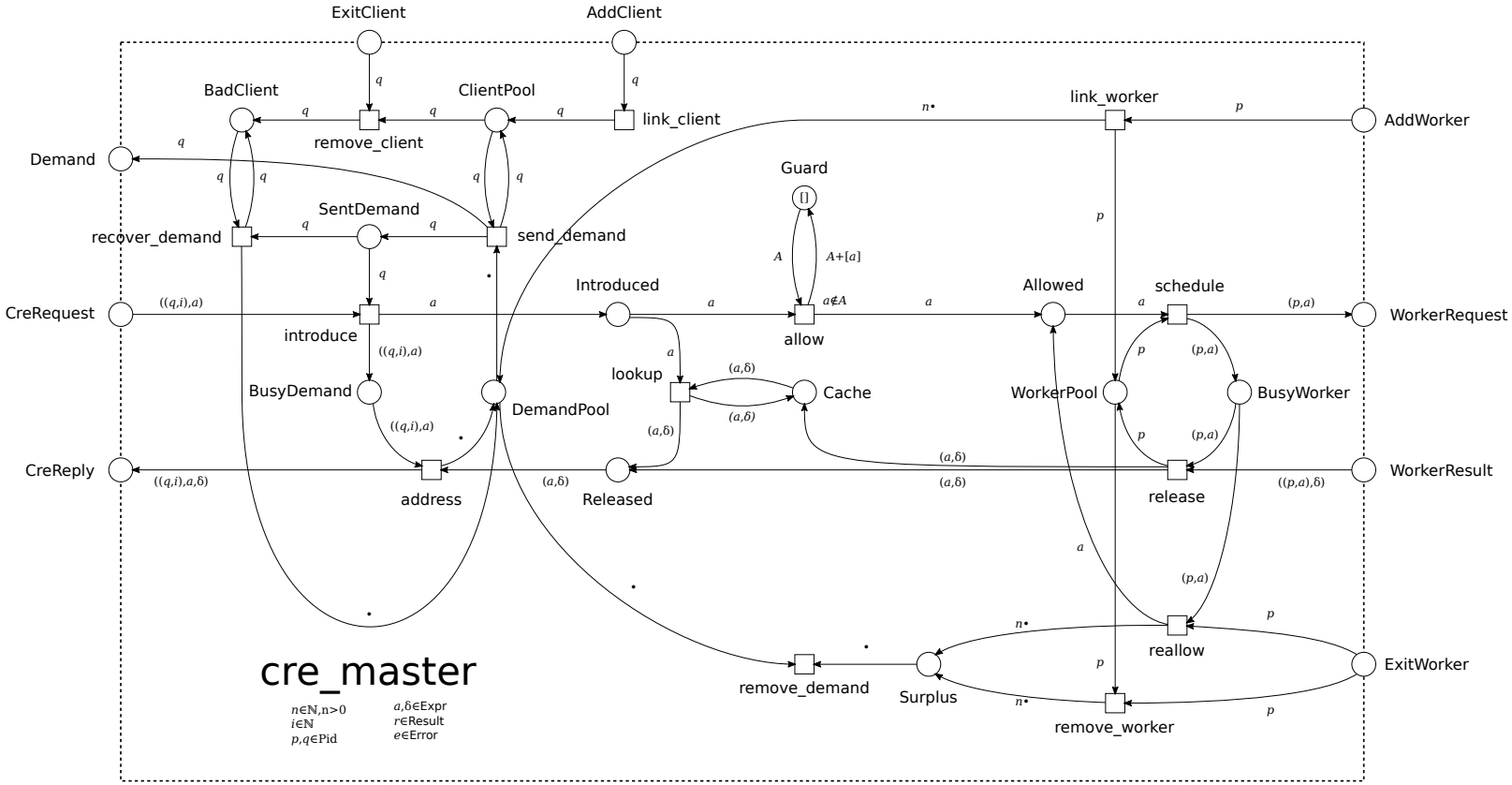
Distributed Execution Environment: Master (i)



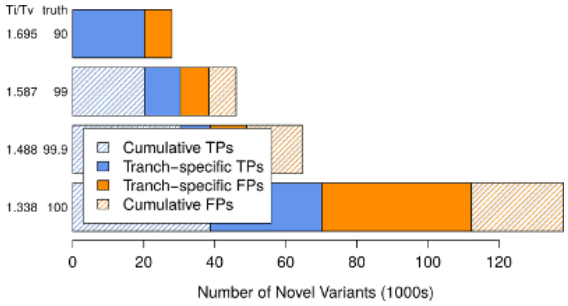
CRE master features

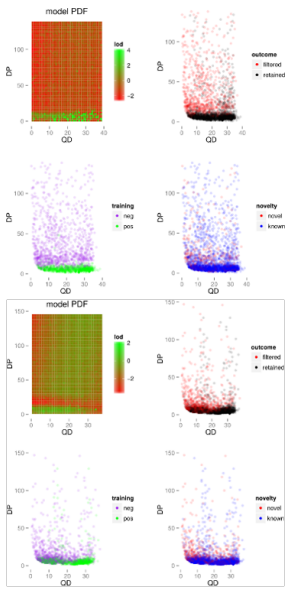
- Cache
- Scheduler
- Fault tolerance
- Language-independence

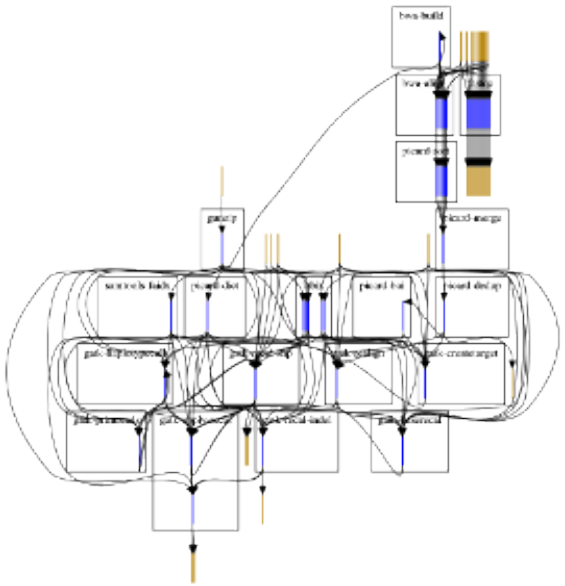
Distributed Execution Environment: Master (ii)

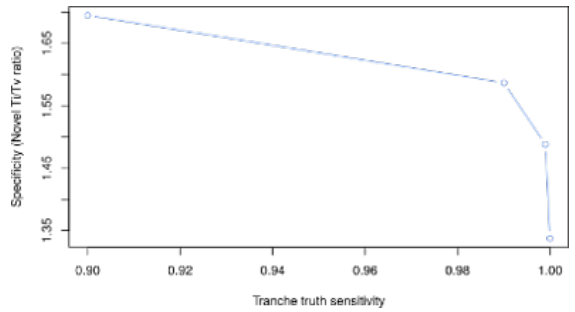


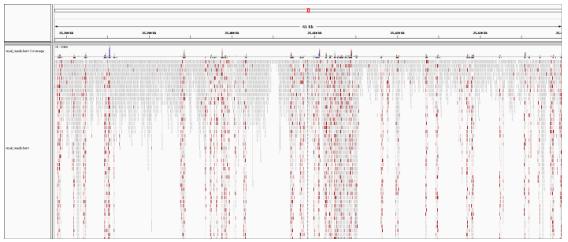
Cuneiform Application: Variant Calling with GATK











https://cuneiform-lang.org

Wrap up

Related Work

- `CPNTools`: Graphical editor with code generation to Erlang
- `Haskell-colored Petri nets`: Functional embeddings

Conclusion

- Petri nets as a programming model
- Visual interpretation of code
- Composition to form distributed systems
  - Petri net on the inside
  - Racket place on the outside

`https://github.com/joergen7/pnet`