Concurrent Programming from pseuCo to Petri

Felix Freiberger, Holger Hermanns
Saarland University

The Problem:

The Problem:
Teaching Concurrency to Students

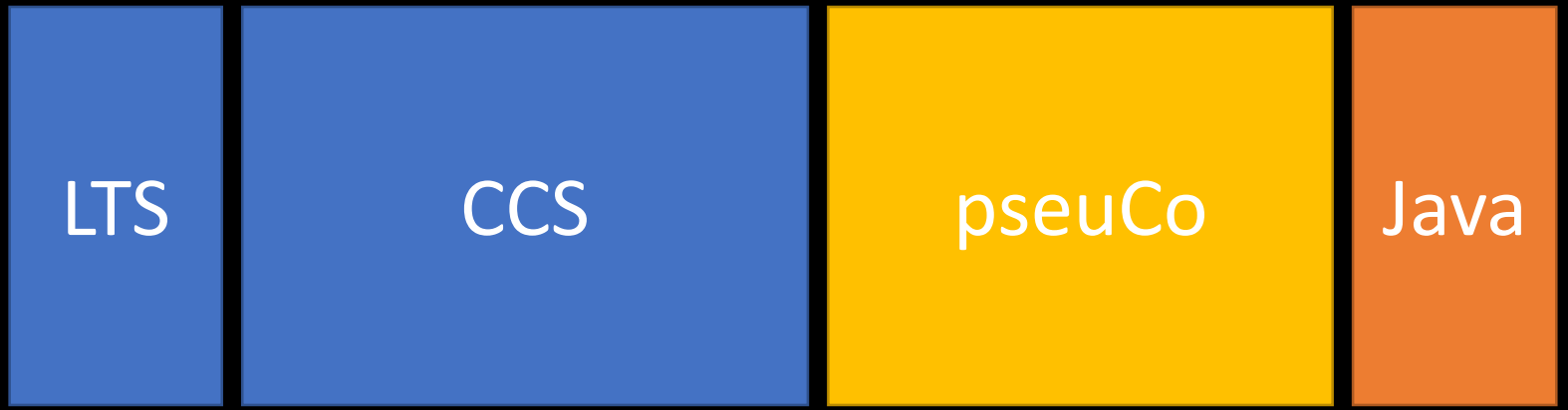
Ye Olde Concurrent Programming Lecture

— est. 2005 —

1. LTS: Labeled Transition Systems
2. CCS: Calculus of Communicating Systems
3. Concurrency in Java

The Concurrent Programming Lecture before 2014

The Concurrent Programming Lecture since 2014



pseuCo: Bridging the Gap from Theory to Practical Programming since 2014

- minimal programming language
- Java-inspired Syntax
- both shared memory and message passing

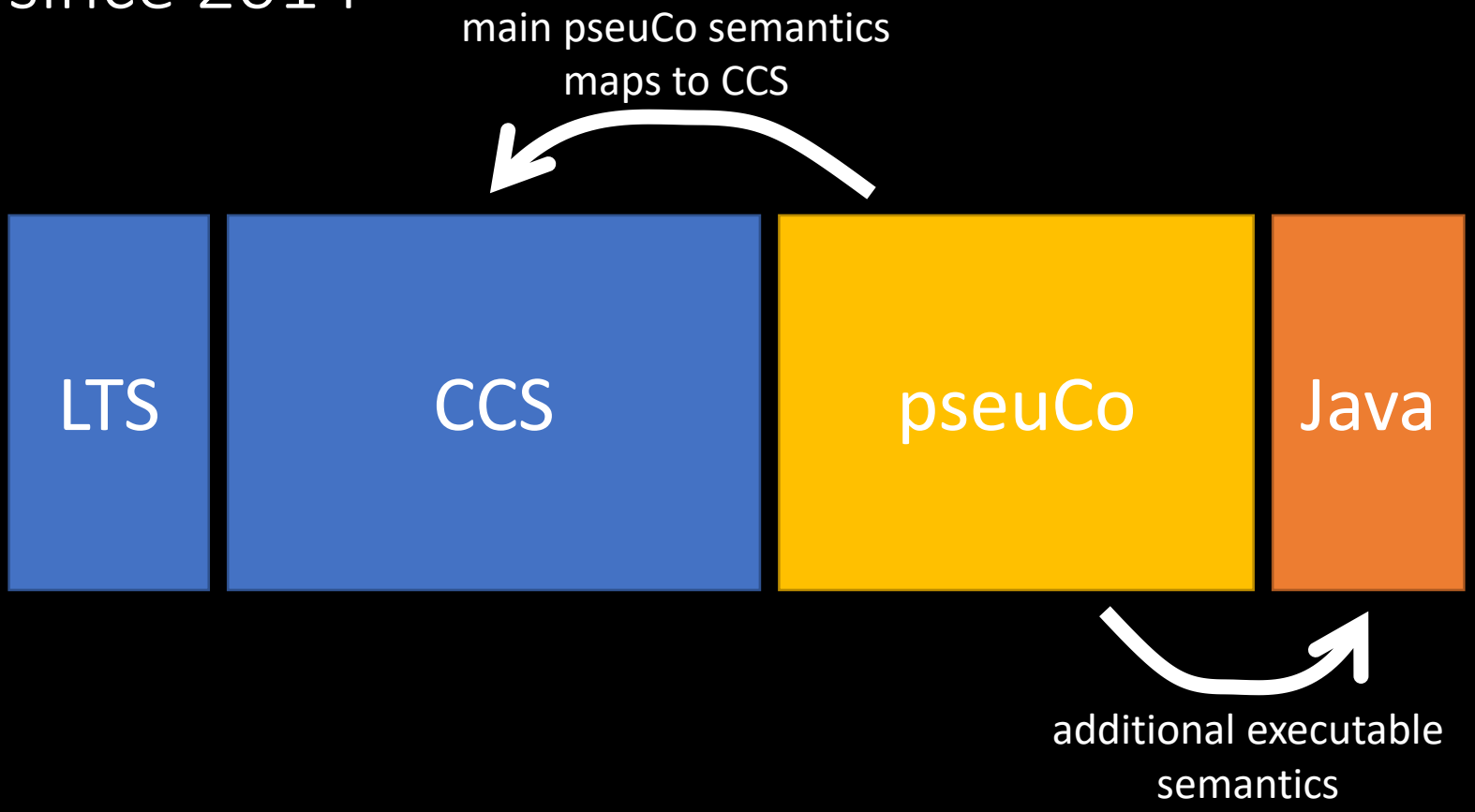
```
void factorial(intchan c) {
    int z, j, n;
    while (true) {
        z = <? c; // receive input

        n = 1;
        for (j = z; j > 0 ; j--) {
            n= n*j;
        }

        c <! n; // send result
    };
}
```

pseuCo by Example

```
void factorial(intchan c) {
    int z, j, n;
    while (true) {
        z = <? c; // receive input

        n = 1;
        for (j = z; j > 0 ; j--) {
            n= n*j;
        }

        c <! n; // send result
    };
}
```

```
mainAgent {
    intchan cc;
    agent a = start(factorial(cc));
    cc <! 3;
    int mid = <? cc;
    println("3! evaluates to " + mid + ".");
    cc <! mid;
    println("(3!)! evaluates to " + (<? cc) + ".");
}
```

The Concurrent Programming Lecture since 2014



Editing First Message Passing pseuCo Duplicate

pseuCo → CCS → LTS Actions

pseuCo

1

No issues.

CCS

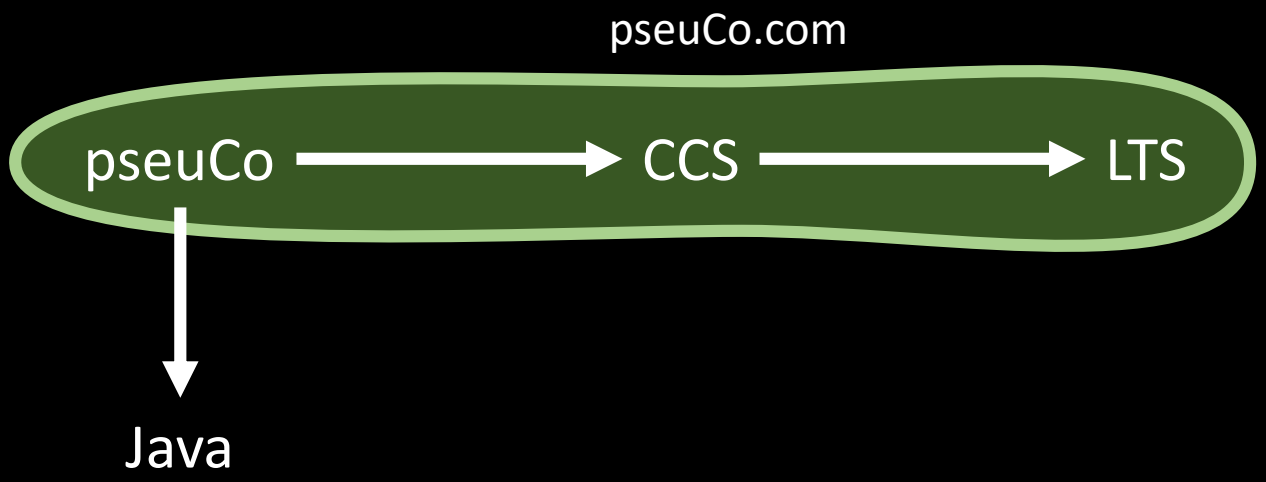
1

No issues.

LTS

Expand all

Where We Are



Contribution of this Paper: Beyond the pseuCo → CCS Semantics

The pseuCo → CCS translation has served us well, but it has problems:

pseuCo → CCS by example

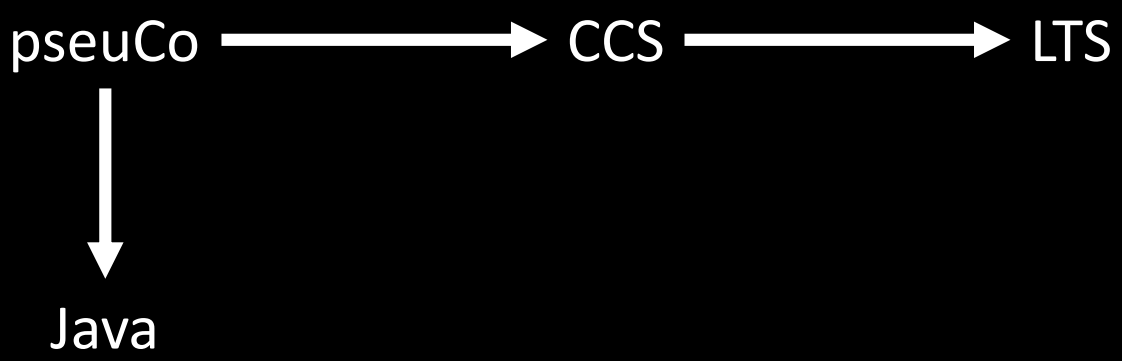
```
Channel_cons[i] := channel_create!(i).(Channel_cons[i-1])

AgentManager[next_i] := agent_new!(next_i).(AgentManager[next_i+1])

Agent_factorial :=
agent_new?i.(start_factorial!(i).(start_set_arg(i)?starter.(start_set_arg(i)?a1.(Agent_factorial |
(Proc_factorial[i, a1] ; agent_terminate(i)!.(0))))))

Proc_factorial[a, $c] := Proc_factorial_1[a, $c, 0, 0, 0]

Proc_factorial_1[a, $c, $z, $j, $n] := when (true) (τ.(receive($c)?$0.(Proc_factorial_3[a, $c, $0, $0, 1]))) +
when (!true) (Proc_factorial_6[a, $c, $z, $j, $n])

Proc_factorial_2[a, $c, $z, $j, $n] := Proc_factorial_3[a, $c, $z, $j-1, $n]

Proc_factorial_3[a, $c, $z, $j, $n] := when (!($j>0)) (Proc_factorial_4[a, $c, $z, $j, $n]) + when ($j>0)
(τ.(Proc_factorial_2[a, $c, $z, $j, $n*$j]))

Proc_factorial_4[a, $c, $z, $j, $n] := when ($c>=0) (put($c)!($n).(Proc_factorial_5[a, $c, $z, $j, $n])) + when
($c<0) (receive($c)!($n).(Proc_factorial_5[a, $c, $z, $j, $n]))

Proc_factorial_5[a, $c, $z, $j, $n] := Proc_factorial_1[a, $c, $z, $j, $n]

Proc_factorial_6[a, $c, $z, $j, $n] := 1

MainAgent[a] := channel_create?$0.(start_factorial?$1.(start_set_arg($1)!(a).(start_set_arg($1)!($0).(when
($0>=0) (put($0)!(3).(MainAgent_1[a, $0, $1])) + when ($0<0) (receive($0)!(3).(MainAgent_1[a, $0, $1]))))))

MainAgent_1[a, $cc, $a] := receive($cc)?$0.(println!("3! evaluates to "^$0^".").(when ($cc>=0)
(put($cc)!($0).(MainAgent_2[a, $cc, $a, $0])) + when ($cc<0) (receive($cc)!($0).(MainAgent_2[a, $cc, $a, $0]))))

MainAgent_2[a, $cc, $a, $mid] := receive($cc)?$0.(println!("(3!)! evaluates to "^$0^".").(0))

(Agent_factorial | MainAgent[1] | Channel_cons[-1] | AgentManager[2]) \ {*, println, exception}
```

Contribution of this Paper: Fixing the pseuCo → CCS semantics

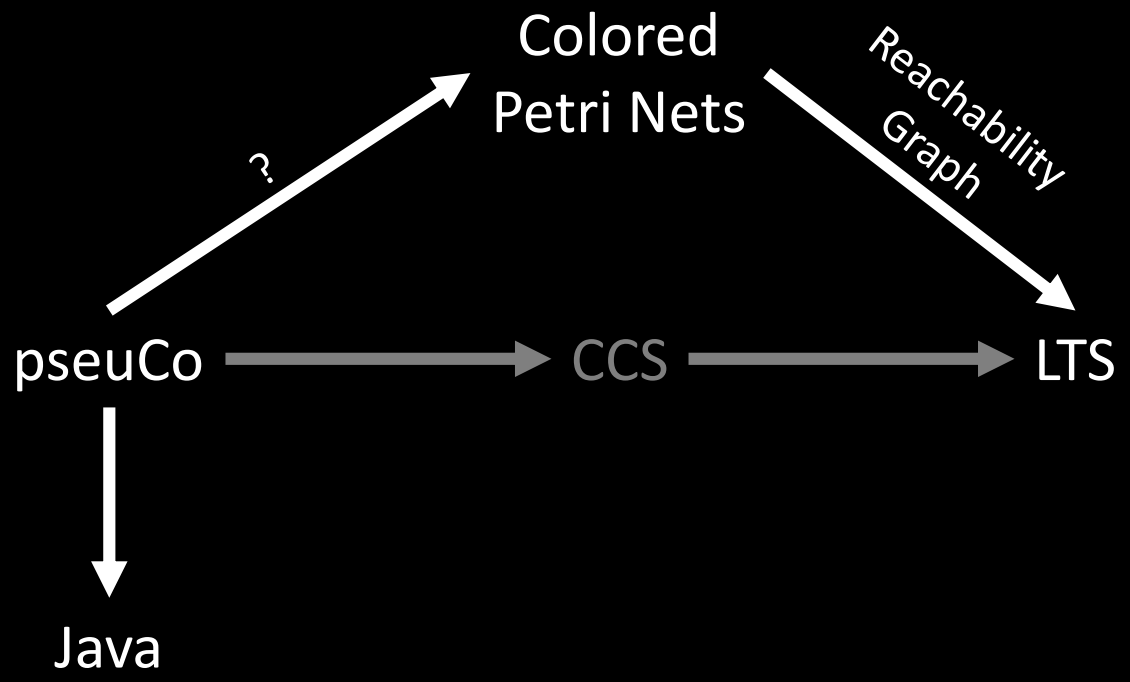
The pseuCo → CCS translation has served us well, but it has problems:

- hard to understand
  - control flow hard to see (goto-style spaghetti code)
  - helper constructs (agent management, channels, arrays, …)
  - low-level hacks visible in the code (e.g. for channels)
- no proper debugging support in pseuCo.com
- no true concurrency due to CCS interleaving semantics
- …and lack of Petri Nets!

Where We Are



Where We Want to Go

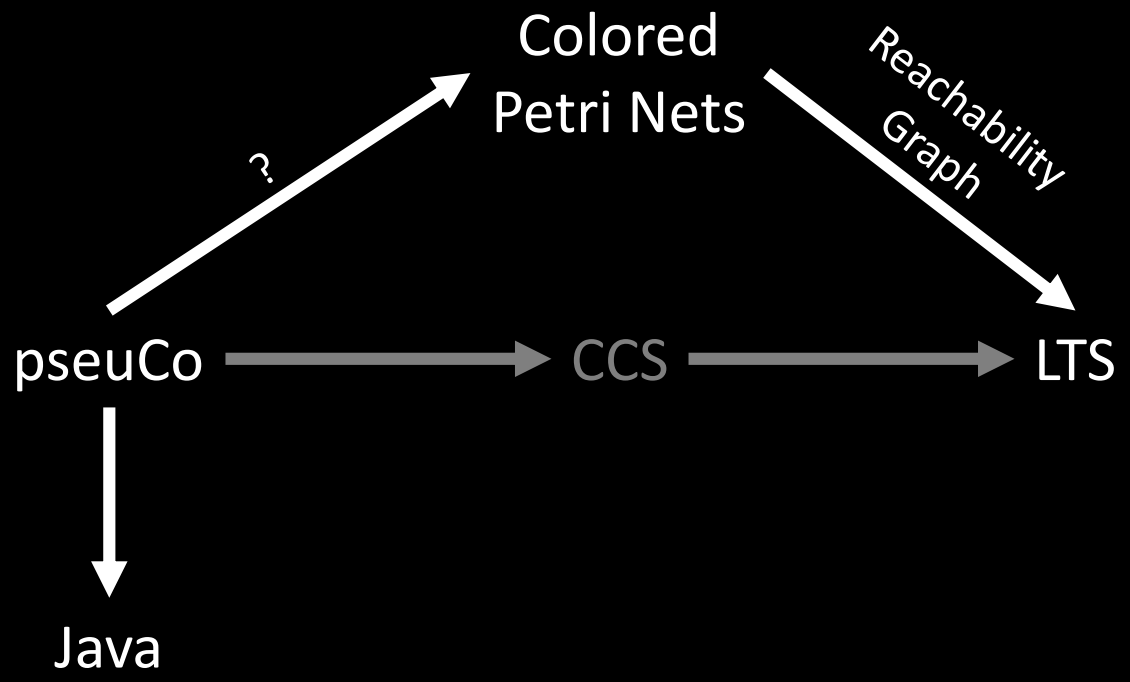


Petri Nets to the Rescue

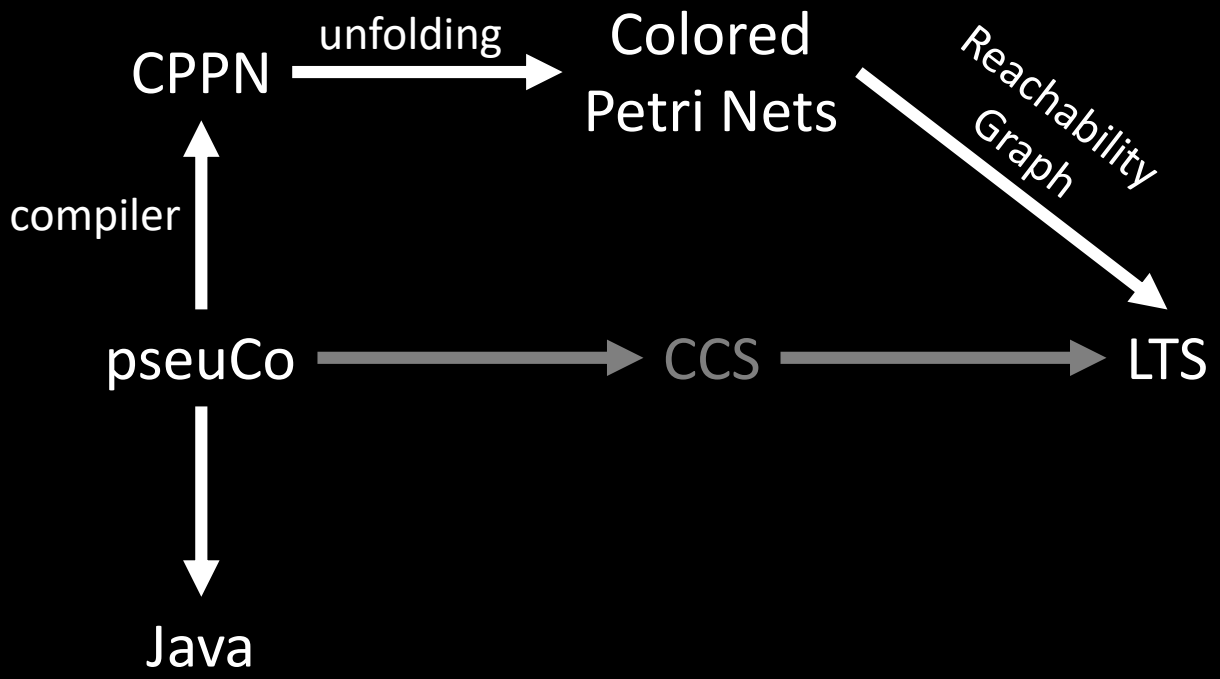
The pseuCo → CCS translation has served us well, but it has problems:

- hard to understand
  - control flow hard to see (goto-style spaghetti code) ✓
  - helper constructs (agent management, channels, arrays, …) ←
  - low-level hacks visible in the code (e.g. for channels) ←
- no proper debugging support in pseuCo.com
- no true concurrency due to CCS interleaving semantics ✓
- …and lack of Petri Nets! ✓

Where We Want to Go



Introducing Colored Program Petri Nets

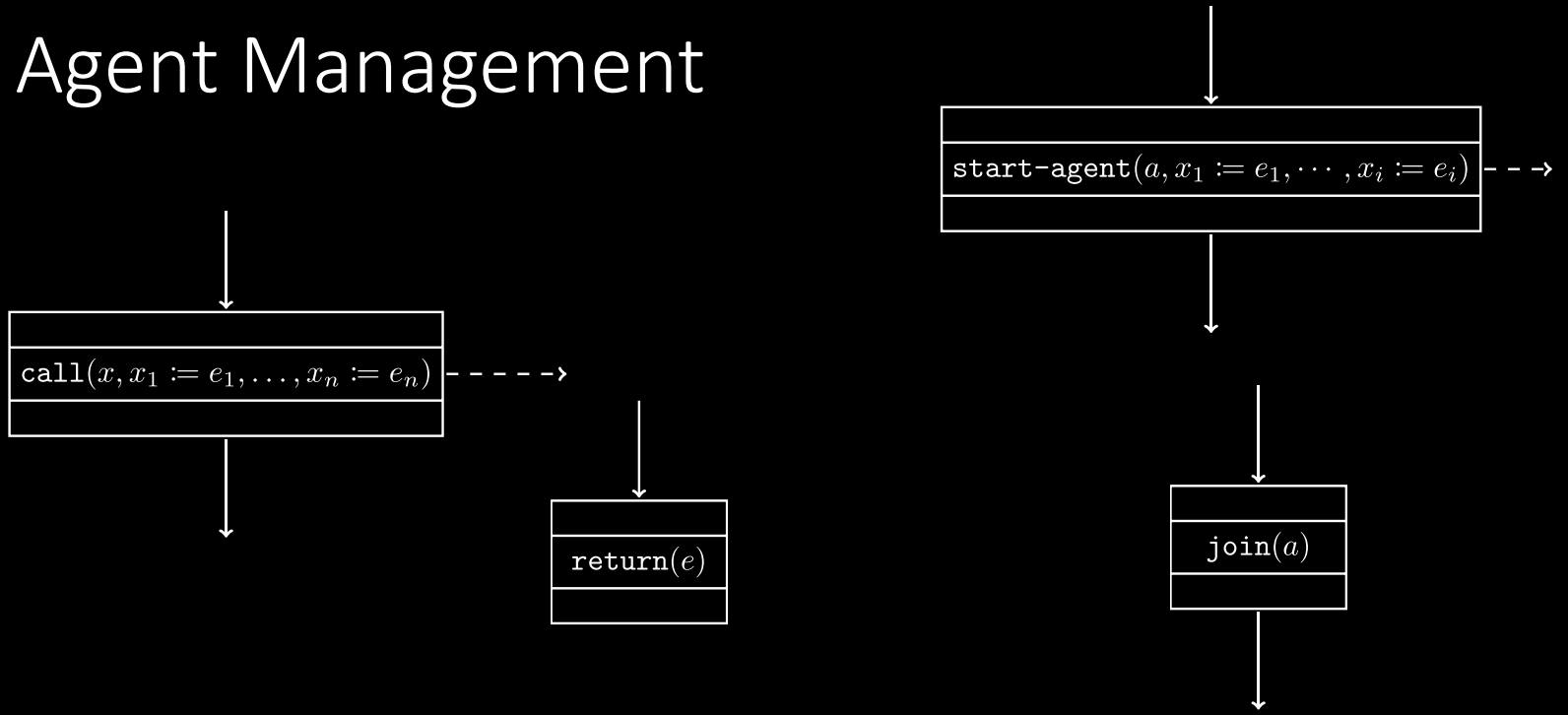


Introducing CPPN: Syntax

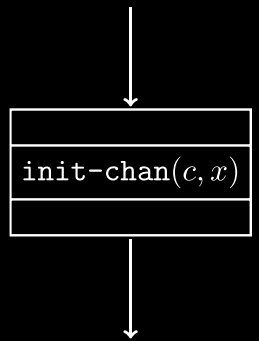
Syntax similar to CPN, but with two kinds of transitions:

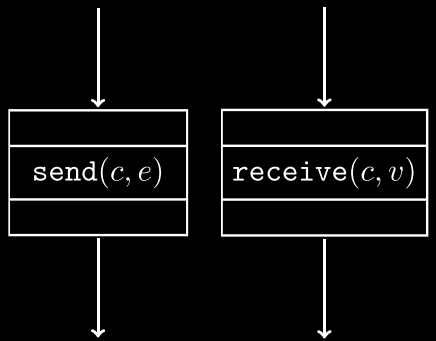
- normal transitions (“Petri transitions”)
- *command* transitions
  - annotated with a command
  - fixed number of arcs

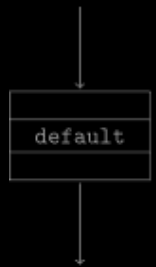
Command Transitions: Agent Management



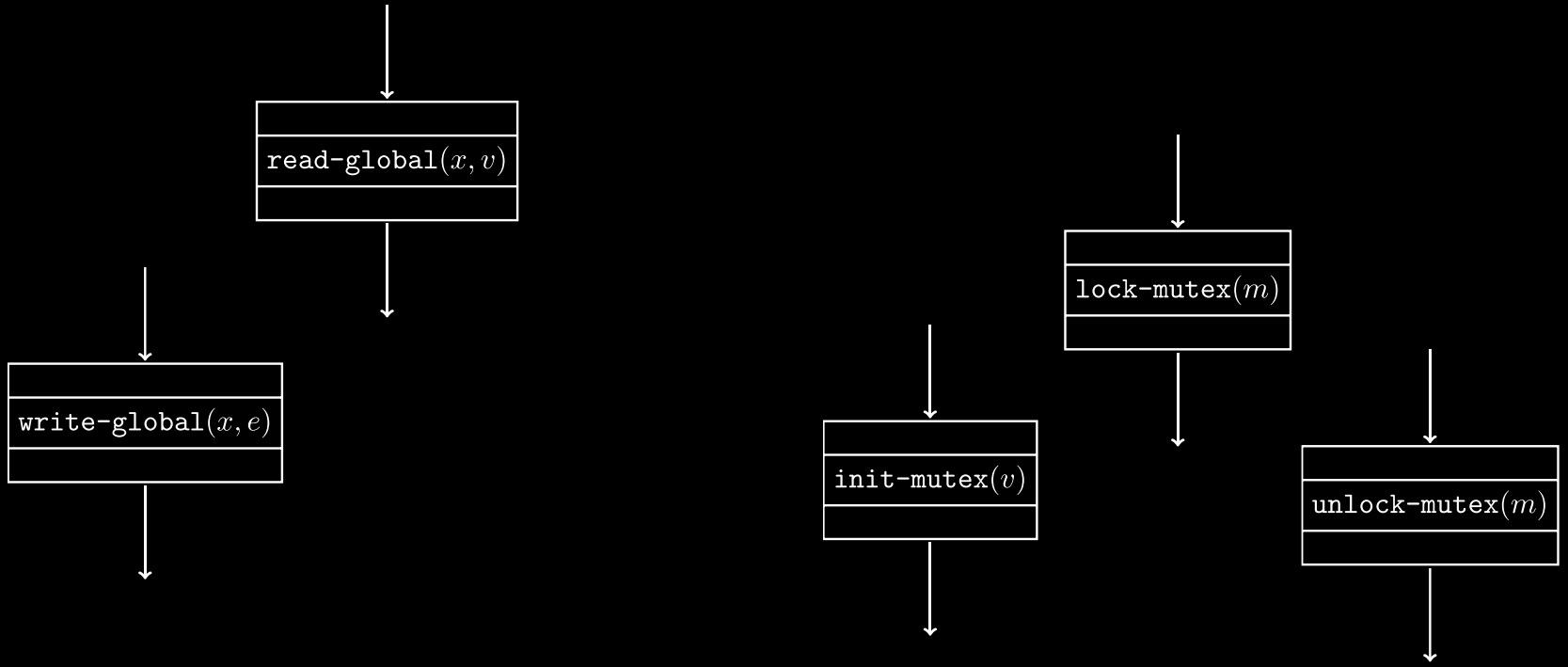
Command Transitions: Message Passing







Command Transitions: Shared Memory

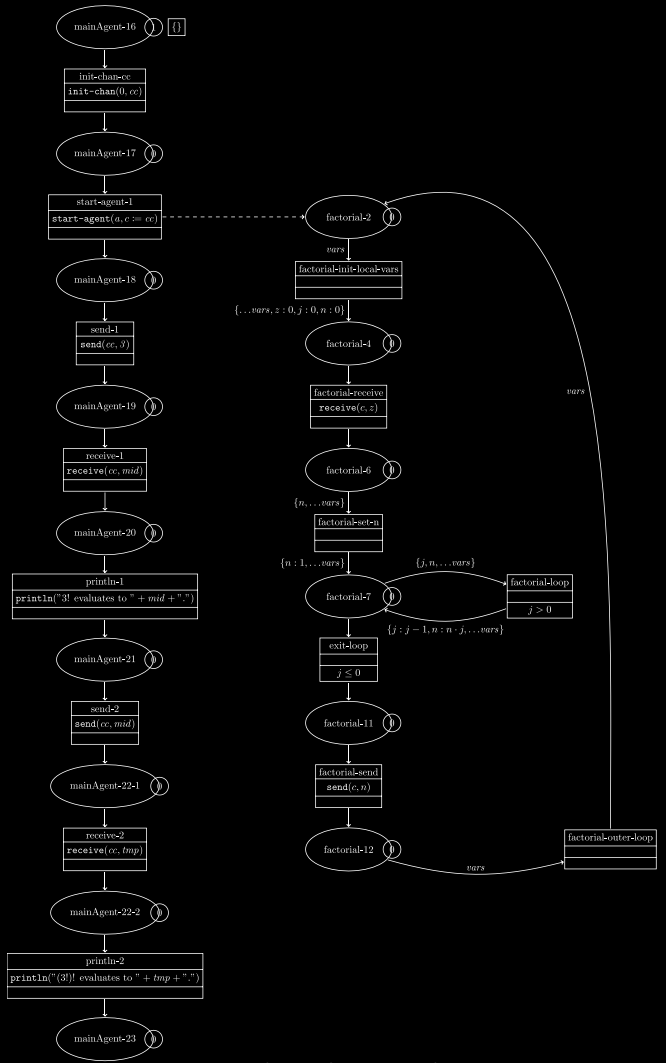


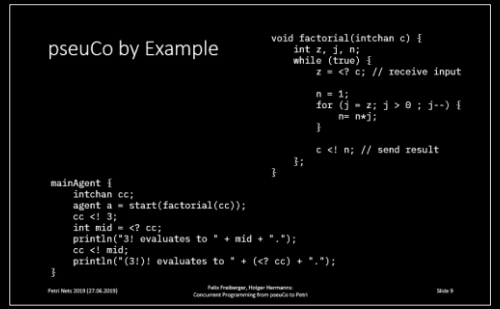
CPPN Syntax restrictions

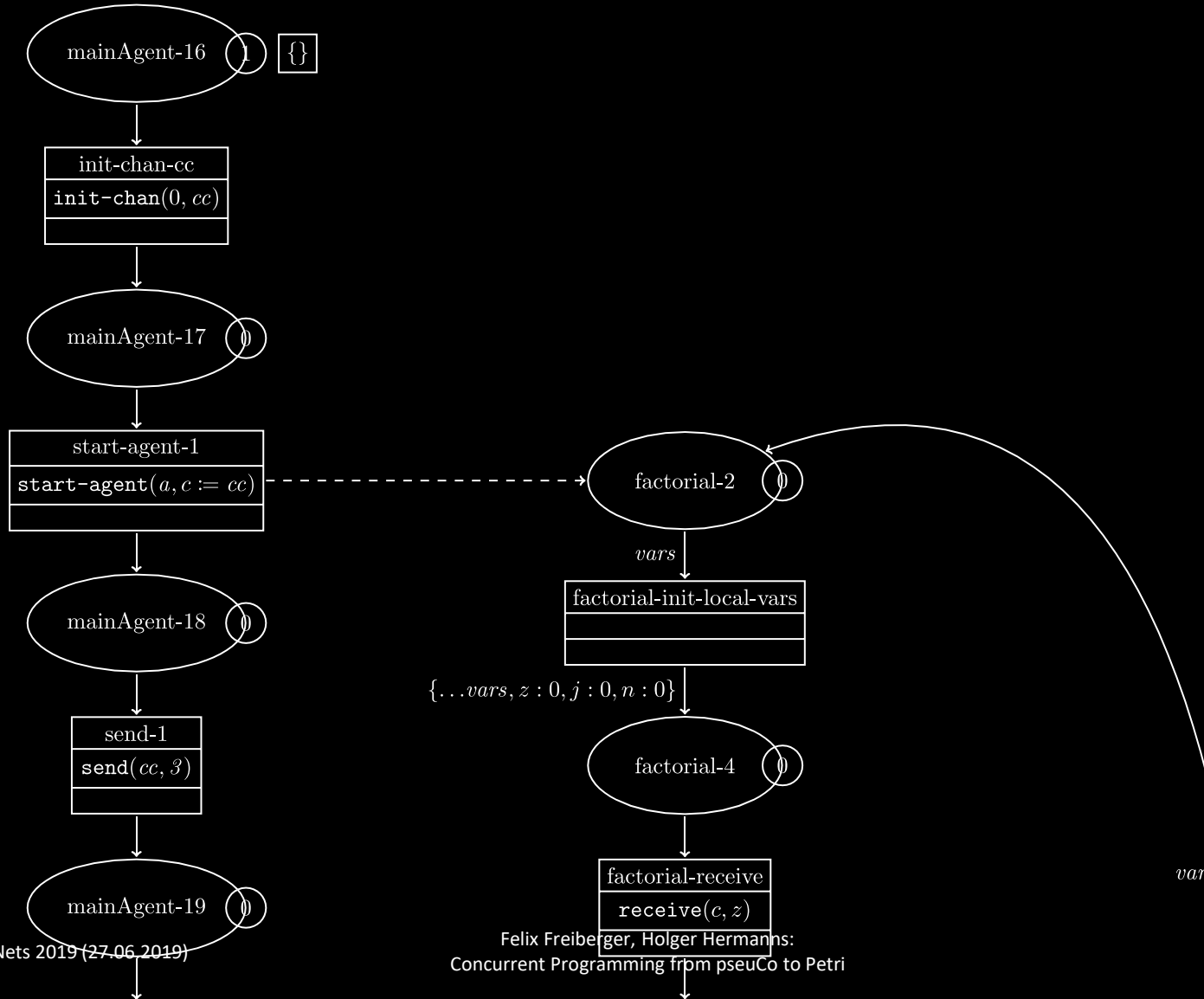
All agent management must be done through the appropriate command transitions:

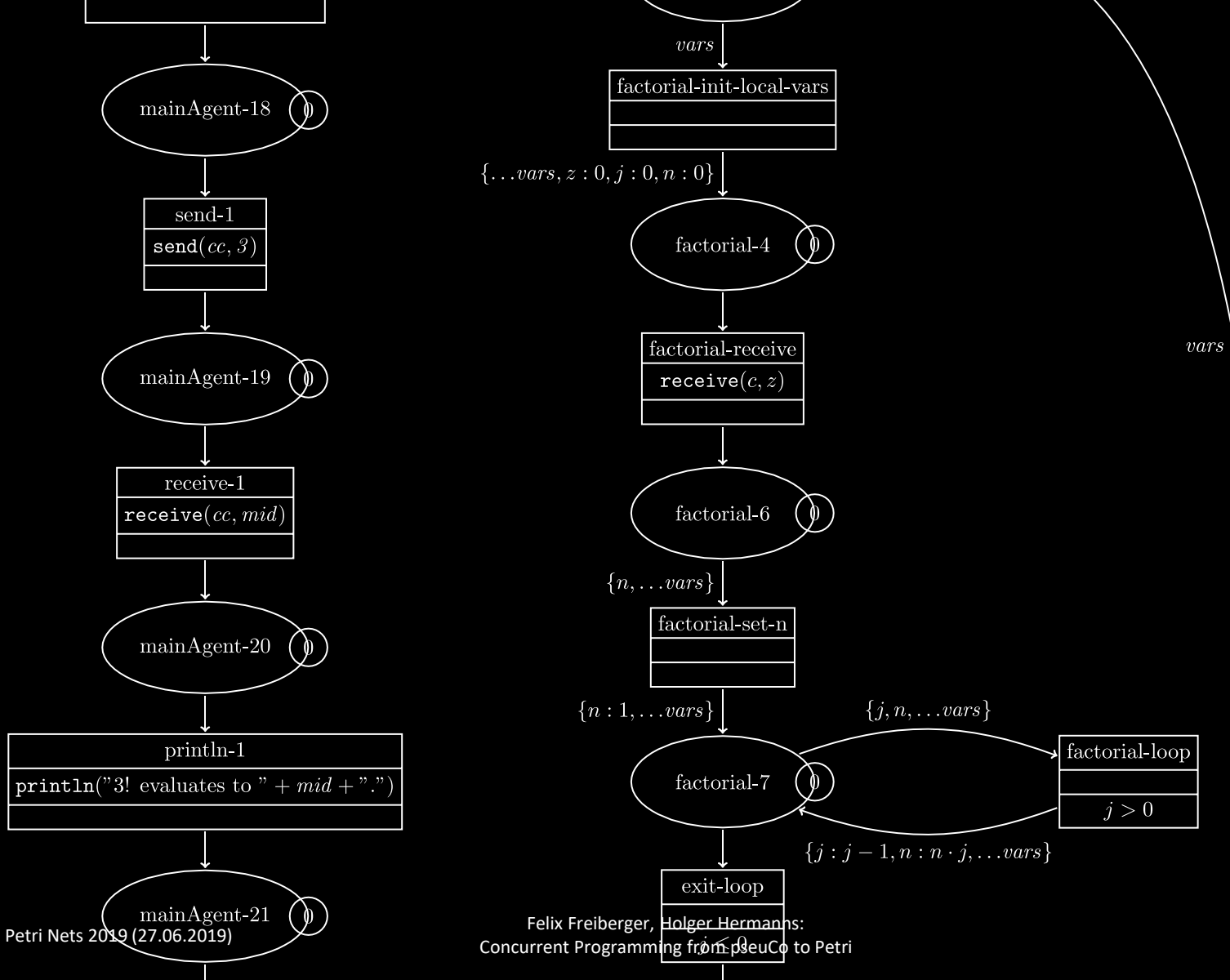
- Petri transitions must have exactly one incoming and outgoing arc
- initial marking must have exactly one token

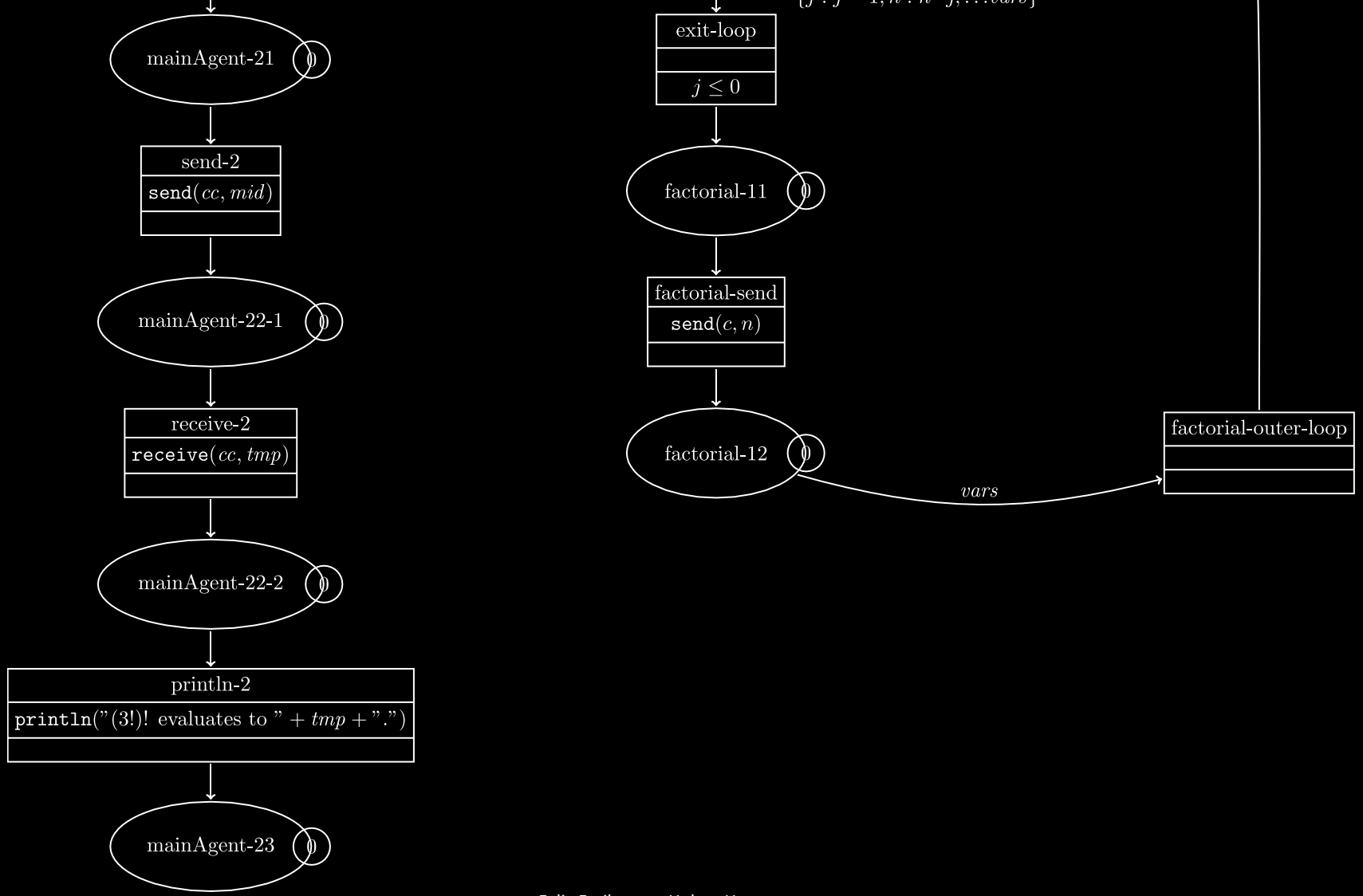
Example


mainAgent-16
{}
init-chan-cc
init-chan(0, cc)
mainAgent-17
start-agent-1
start-agent(a, c := cc)
factorial-2
vars
factorial-init-local-vars
{...vars, z : 0, j : 0, n : 0}
mainAgent-18
factorial-1
send-1
send(cc, 3)
mainAgent-19
factorial-receive
receive(c, z)
vars
receive-1
receive(cc, mid)
factorial-6
{n, ...vars}
mainAgent-20
factorial-set-n
println-1
println("3! evaluates to " + mid + ".")
{n : 1, ...vars}
{j, n, ...vars}
factorial-7
factorial-loop
j > 0
{j : j − 1, n : n · j, ...vars}
mainAgent-21
exit-loop
j ≤ 0
send-2
send(cc, mid)
factorial-11
mainAgent-22-1
factorial-send
send(c, n)
receive-2
receive(cc, tmp)
factorial-12
factorial-outer-loop
vars
mainAgent-22-2
println-2
println("(3!)! evaluates to " + tmp + ".")
mainAgent-23


pseuCo by Example
void factorial(intchan c) {
int z, j, n;
while (true) {
z = <? c; // receive input
n = 1;
for (j = z; j > 0 ; j--) {
n= n*j;
}
c <! n; // send result
};
}
mainAgent {
intchan cc;
agent a = start(factorial(cc));
cc <! 3;
int mid = <? cc;
println("3! evaluates to " + mid + ".");
cc <! mid;
println("(3!)! evaluates to " + (<? cc) + ".");
}
Petri Nets 2019 (27.06.2019)
Felix Freiberger, Holger Hermanns:
Concurrent Programming from pseuCo to Petri
Slide 9


mainAgent-16
1
{}
init-chan-cc
init-chan(0, cc)
mainAgent-17
0
start-agent-1
start-agent(a, c := cc)
factorial-2
0
vars
mainAgent-18
0
factorial-init-local-vars
{...vars, z : 0, j : 0, n : 0}
send-1
send(cc, 3)
factorial-4
0
mainAgent-19
0
factorial-receive
receive(c, z)
vars


mainAgent-18
send-1
send(cc, 3)
mainAgent-19
receive-1
receive(cc, mid)
mainAgent-20
println-1
println("3! evaluates to " + mid + ".")
mainAgent-21
vars
factorial-init-local-vars
{...vars, z : 0, j : 0, n : 0}
factorial-4
factorial-receive
receive(c, z)
factorial-6
{n, ...vars}
factorial-set-n
{n : 1, ...vars}
{j, n, ...vars}
factorial-7
factorial-loop
j > 0
{j : j − 1, n : n · j, ...vars}
exit-loop
j < 0
vars


mainAgent-21
send-2
send(cc, mid)
mainAgent-22-1
receive-2
receive(cc, tmp)
mainAgent-22-2
println-2
println("(3!)! evaluates to " + tmp + ".")
mainAgent-23
exit-loop
j ≤ 0
factorial-11
factorial-send
send(c, n)
factorial-12
vars
factorial-outer-loop

Semantics

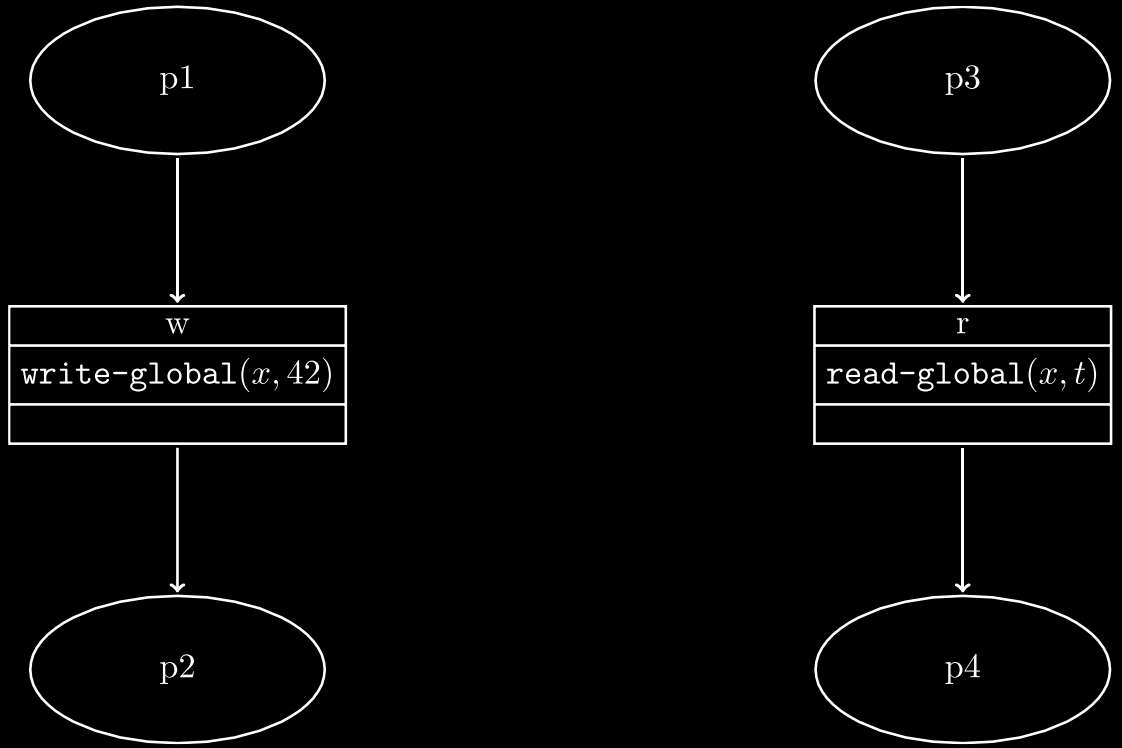
CPPNs unfold to regular CPNs.

This unfolding…

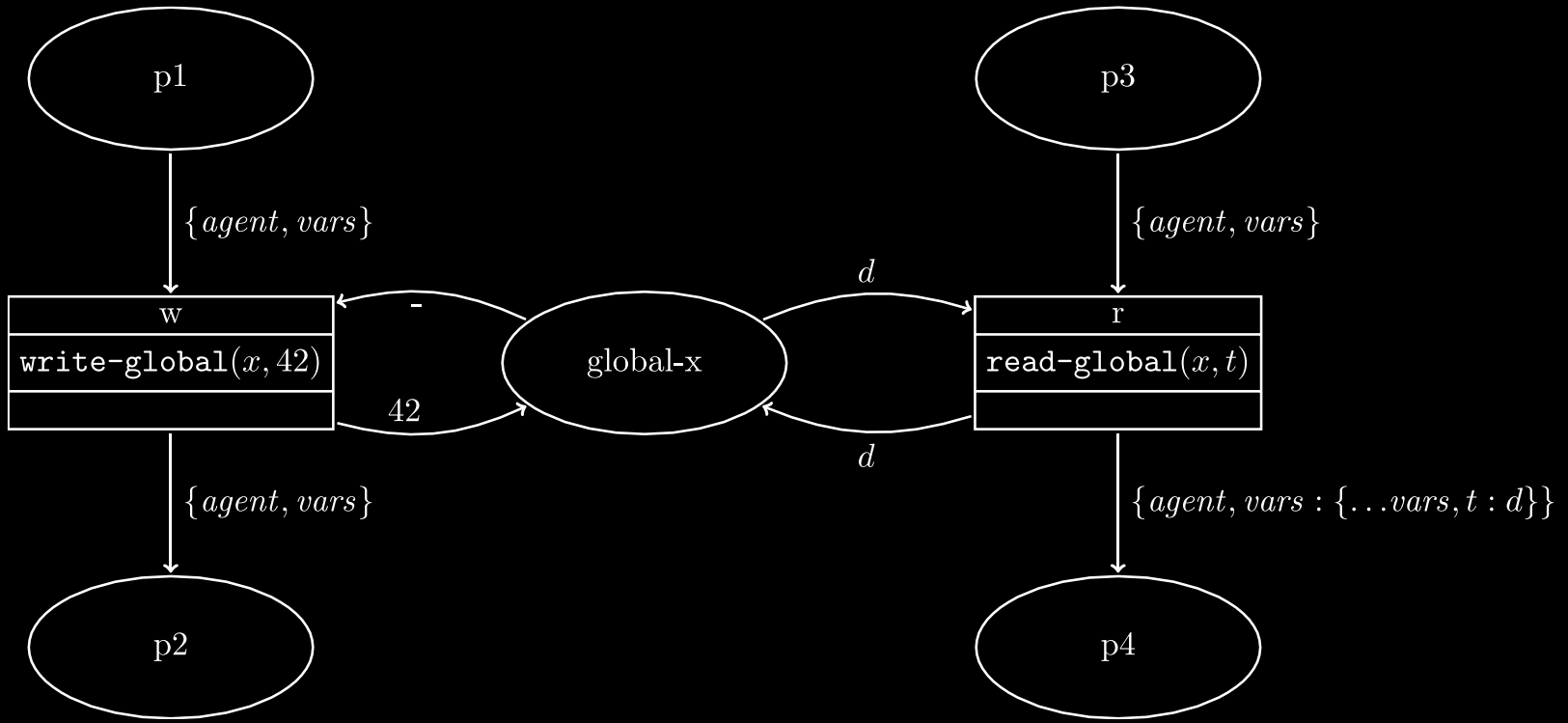
- adds agent identifiers to each token
- replaces most command transitions with fixed static constructs
- inserts transitions linking `call` ↔ `return` and `send` ↔ `receive`

Complexity: $\mathcal{O}(n^2)$ in number of these transitions, $\mathcal{O}(n)$ for everything else

Unfolding Example

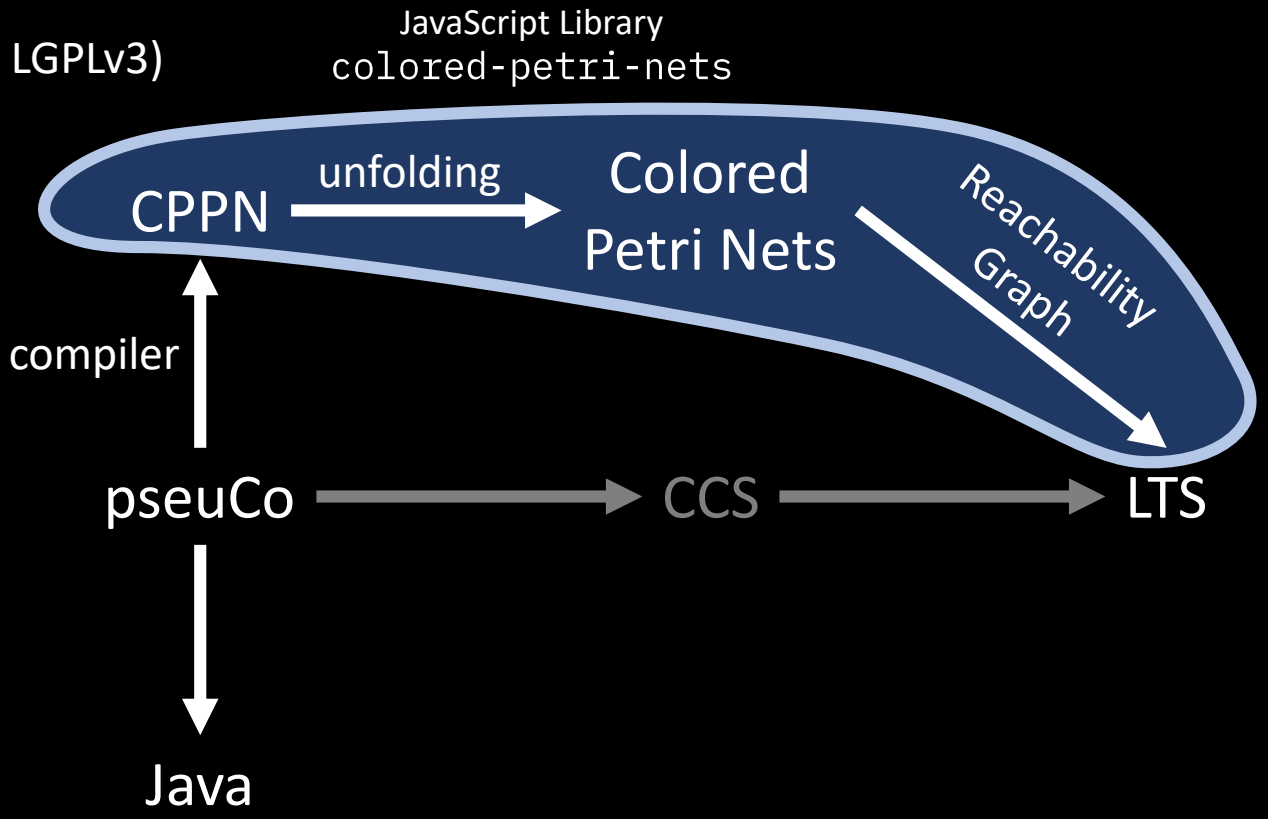


Unfolding Example



Implementation

- published on NPM
- open source (GNU LGPLv3)

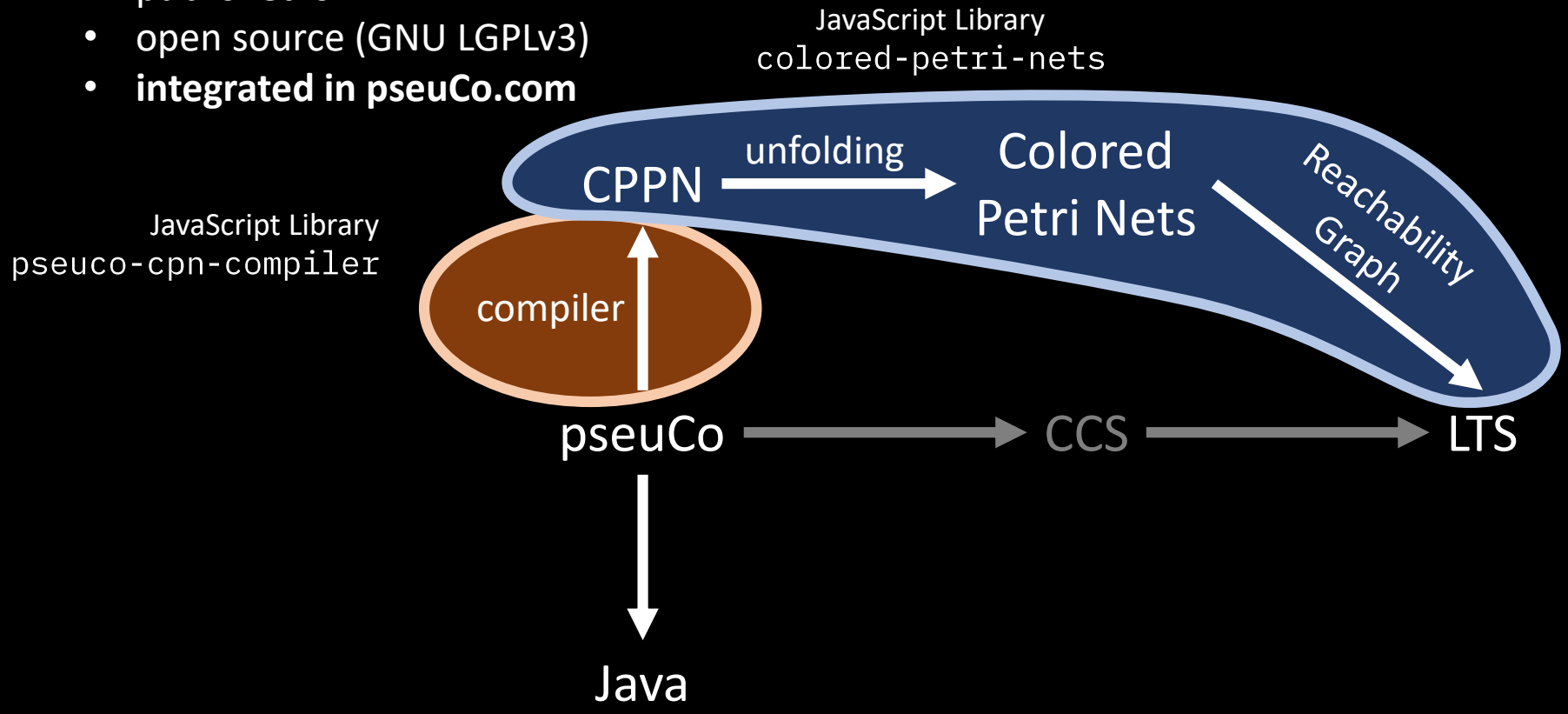


Compiling pseuCo to CPPN

- straightforward compiler
- translates AST nodes into CPPN transitions
- about 2.500 lines of code, with full support for
  - channels
  - select-case
  - global variables
  - locks
  - arrays new
  - structures new
  - monitors new
  - condition synchronization new

Implementation

- published on NPM
- open source (GNU LGPLv3)
- **integrated in pseuCo.com**



pseuCo.com Exercises Files Debug Backup Help About

Editing First Message Passing pseuCo Duplicate

pseuCo → CCS → LTS Actions

pseuCo

```
1  void factorial(intchan c) {
2      int z, j, n;
3      while (true) {
4          z = <? c; // receive input
5
6          n = 1;
7          for (j = z; j > 0; j--) {
8              n= n*j;
9          }
10
11         c <! n; // send result
12     };
13 }
14
15 mainAgent {
16     intchan cc;
17     agent a = start(factorial(cc));
18     cc <! 3;
19     int mid = <? cc;
20     println("3! evaluates to " + mid + ".");
21     cc <! mid;
22     println("(3!)! evaluates to " + (<?
   cc) + ".");
23 }
24
```

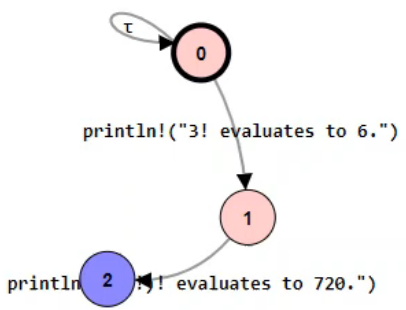
No issues.

CCS

```
1  Channel_cons[i] := channel_create!(i).
   (Channel_cons[i-1])
2  AgentManager[next_i] := agent_new!(next_i).
   (AgentManager[next_i+1])
3  Agent_factorial := agent_new?i.
   (start_factorial!(i).(start_set_arg(i)?
   starter.(start_set_arg(i)?a1.
   (Agent_factorial |
   (Proc_factorial[i, a1] ; agent_terminate(i)!.
   (0))))))
4
5  Proc_factorial[a, $c] := Proc_factorial_1[a, $
   c, 0, 0, 0]
6  Proc_factorial_1[a, $c, $z, $j, $n] := when (t
   rue) (τ.(receive($c)?$0.
   (Proc_factorial_3[a, $c, $0, $0, 1]))) + when
   (!true) (Proc_factorial_6[a, $c, $z, $j, $n])
7  Proc_factorial_2[a, $c, $z, $j, $n] := Proc_fa
   ctorial_3[a, $c, $z, $j-1, $n]
8  Proc_factorial_3[a, $c, $z, $j, $n] := when (!
   ($j>0)) (Proc_factorial_4[a, $c, $z, $j, $n])
   + when ($j>0) (τ.
   (Proc_factorial_2[a, $c, $z, $j, $n*$j]))
9  Proc_factorial_4[a, $c, $z, $j, $n] := when ($
   c>=0) (put($c)!($n).
   (Proc_factorial_5[a, $c, $z, $j, $n])) + when
   ($c<0) (receive($c)!($n).
   (Proc_factorial_5[a, $c, $z, $j, $n]))
10 Proc_factorial_5[a, $c, $z, $j, $n] := Proc_fa
   ctorial_1[a, $c, $z, $j, $n]
```

No issues.

LTS



Return Collapse all Expand all

Petri Nets to the Rescue

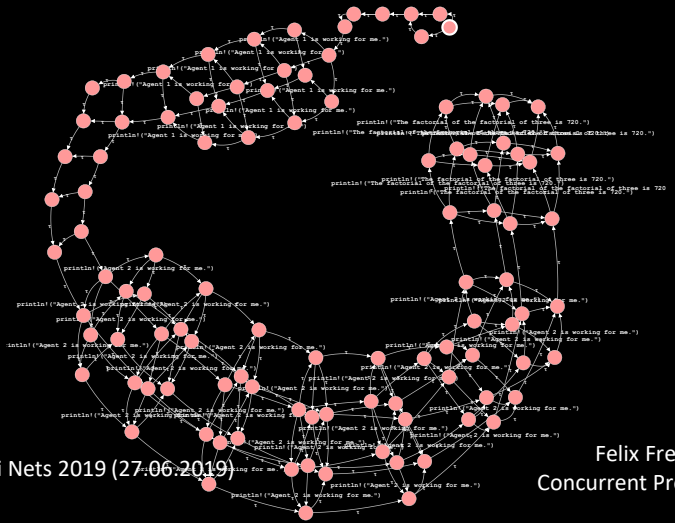
The pseuCo → CCS translation has served us well, but it has problems:

- hard to understand ✓?
  - control flow hard to see (goto-style spaghetti code) ✓
  - helper constructs (agent management, channels, arrays, …) ✓
  - low-level hacks visible in the code (e.g. for channels) ✓
- no proper debugging support in pseuCo.com ⟵
- no true concurrency due to CCS interleaving semantics ✓
- …and lack of Petri Nets! ✓

Debugging pseuCo: Previous Options

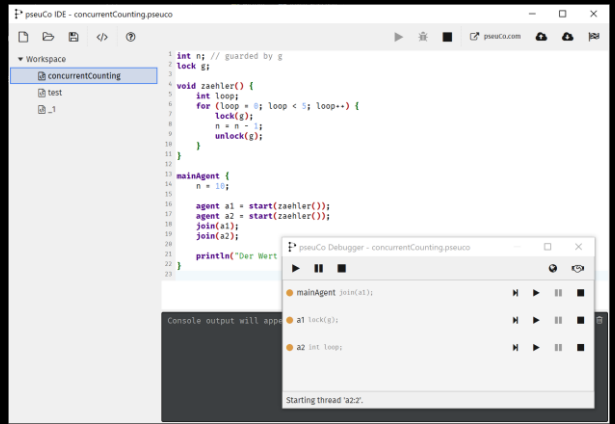
Debugging with CCS and LTS

- nondeterminism fully preserved
- **state identifiers are CCS terms, very hard to understand program state**



Debugging with pseuCo IDE

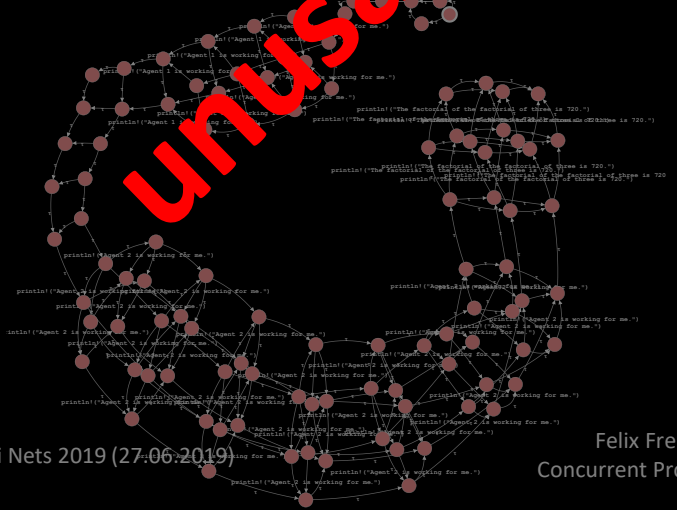
- traditional, IDE-style debugging
- based on pseuCo → Java semantics
- **no control over nondeterminism**



Debugging pseuCo: Previous Options

Debugging with CCS and LTS

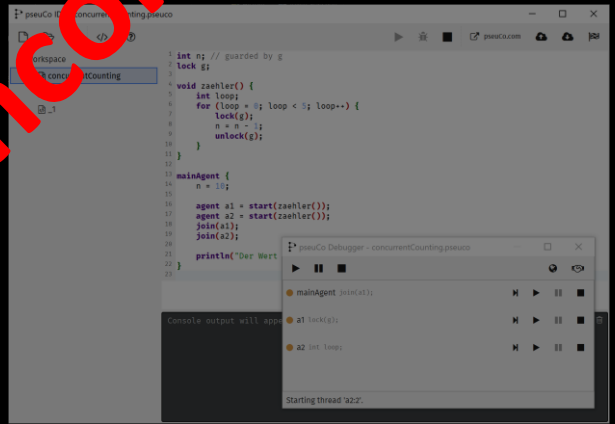
- nondeterminism fully preserved
- **state identifiers are CCS terms, very hard to understand program state**



unusable

Debugging with pseuCo IDE

- traditional, IDE-style debugging
- based on pseuCo → Java semantics
- **no control over nondeterminism**



incomplete

Debugging pseuCo: Using Petri Nets

- in principle, the reachability graph is perfect for debugging
  - nondeterminism fully preserved
  - markings are easier to comprehend than CCS terms (once you have understood the Petri net)
- but it requires understanding of Petri nets and compiler internals

Can we build a debugger that feels like an IDE, but is actually just walking the reachability graph?

pseuCo.com | Exercises | Files | Debug | Backup | Help | About

Editing First Message Passing pseuCo Duplicate

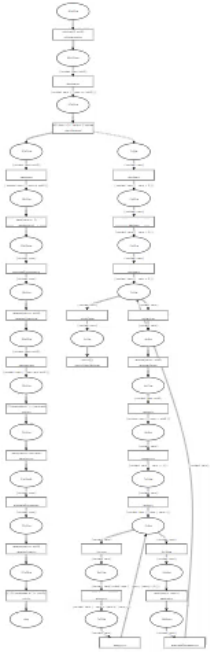
pseuCo → CPPN → CPN → LTS (experimental) | ▶ pseuCo Debugger (experimental)

pseuCo

```
1  void factorial(intchan c) {
2      int z, j, n;
3      while (true) {
4          z = <? c; // receive input
5
6          n = 1;
7          for (j = z; j > 0; j--) {
8              n= n*j;
9          }
10
11         c <! n; // send result
12     };
13 }
14
15 mainAgent {
16     intchan cc;
17     agent a = start(factorial(cc));
18     cc <! 3;
19     int mid = <? cc;
20     println("3!
       evaluates to " + mid + ".");
21     cc <! mid;
22     println("(3!)!
       evaluates to " + (<? cc) + ".");
23 }
24
```

No issues.

CPPN



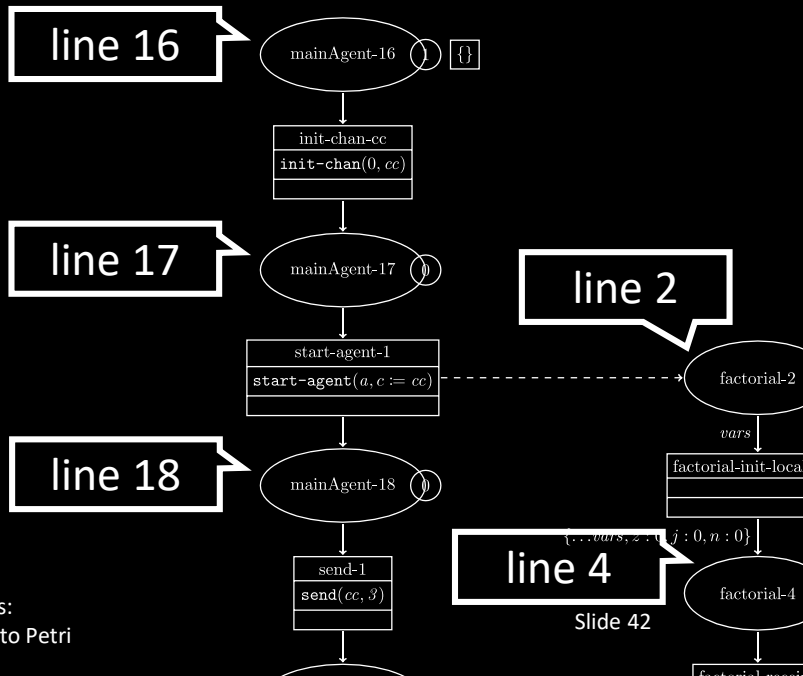
Force Layout

CPN

LTS

pseuCo Debugger Internals

- walks reachability graph
- net annotated with auxiliary information (globally and per place)



Summary

- CPPN: a CPN-like formalism for concurrent programs, designed for teaching
- unfolding to regular CPNs
- pseuCo → CPPN
- implemented and integrated into pseuCo.com
- Debugging support in pseuCo.com

More information at the tool exhibition!