LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Semantics and Analysis of KLAIM Models in Maude*

Martin Wirsing[1]

In cooperation with
Jonas Eckhardt[2], Tobias Mühlbauer[2], José Meseguer[3]

[1]Ludwig-Maximilians-Universität München
[2]Technische Universität München, [3]University of Illinois at Urbana-Champaign
IFIP WG 1.3
Hothorpe Hall, January 2014

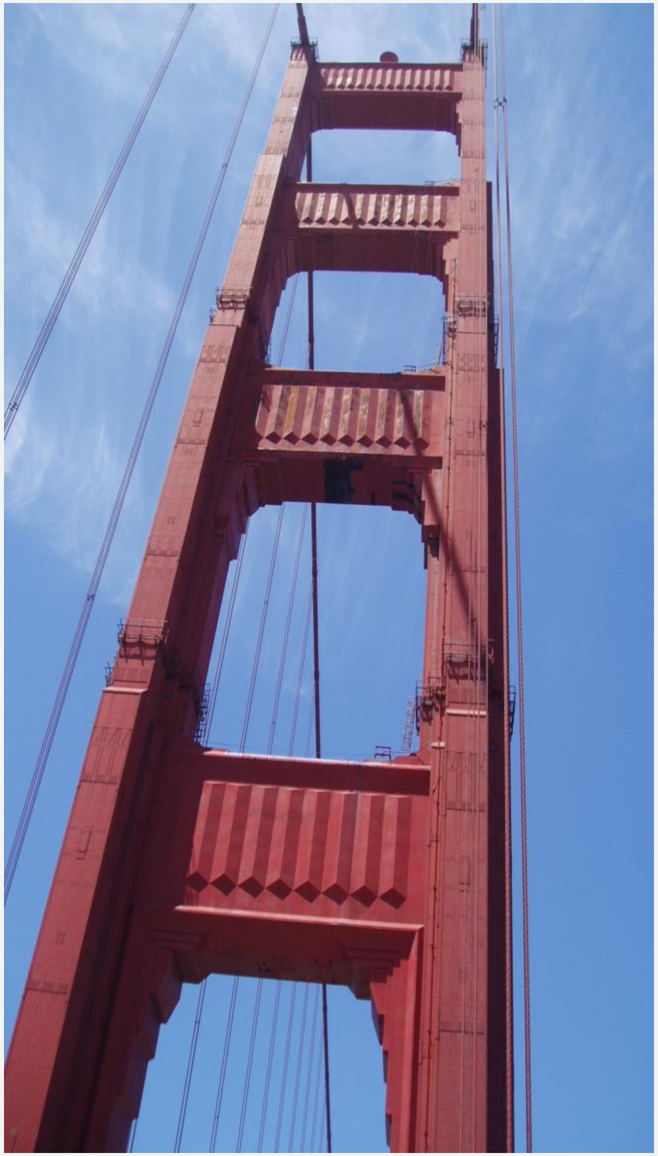
• Partially supported by EU-project ASCENS

Goal and Approach

- **ASCENS Project**
  - Languages, theories and tools for engineering autonomic systems in distributed environments
  - Case studies:
    - Robot swarm, Peer2Peer Cloud, E-mobility

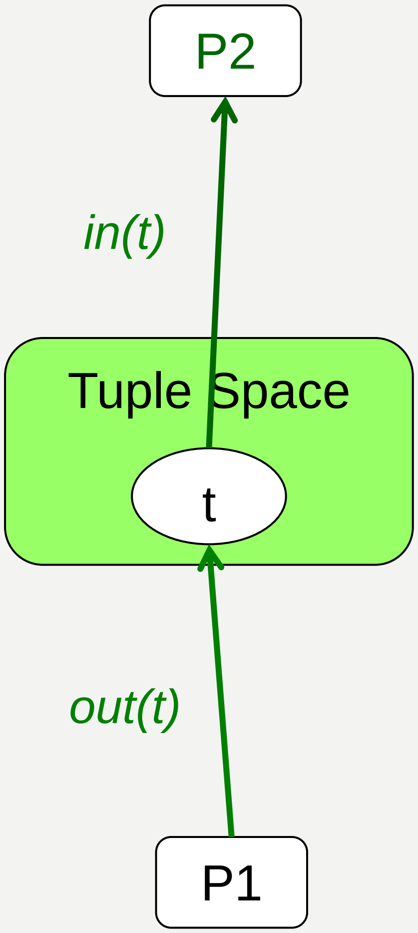
- **Goal of this talk:**
  - Correct simulation and analysis of a specification language for distributed (autonomic) systems

- **Approach:**
  - Choose KLAIM as coordination language
  - Rewriting Logic as a semantic framework
  - Formal analysis using the Maude environment



- **Tuple space coordination model**
  - Linda [Gelernter et al 1985]
    - Tuple space concept
  - **KLAIM [De Nicola et al. 1997]**
    - Distributed tuple space, CCS-like computation
  - SCEL [Pugliese, De Nicola et. al. 2011/13]
    - Distributed tuple space, policy-controlled computation

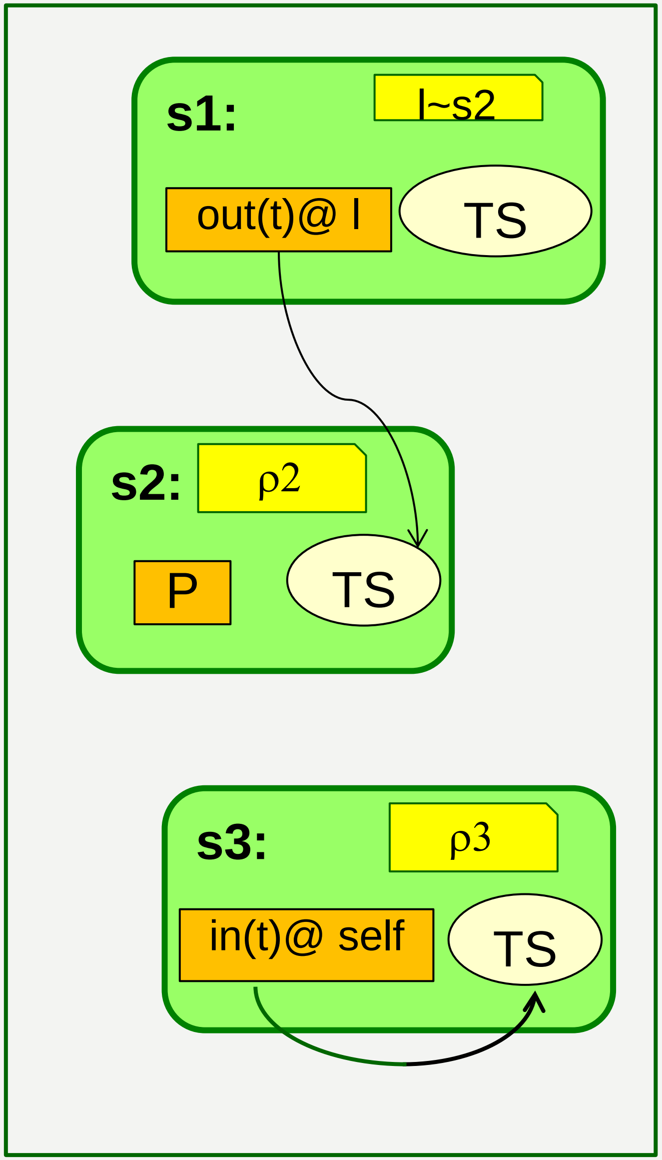
Linda



KLAIM

(**K**ernel **L**anguage for **A**gents **I**nteraction and **M**obility)

- Language for distributed mobile computing
- **KLAIM Structure**
  - Nets are composed of Nodes
  - Nodes have a unique location and contain a CCS-like process
  - Processes reflect the tuple space concept
  - Mobility is modeled by moving processes

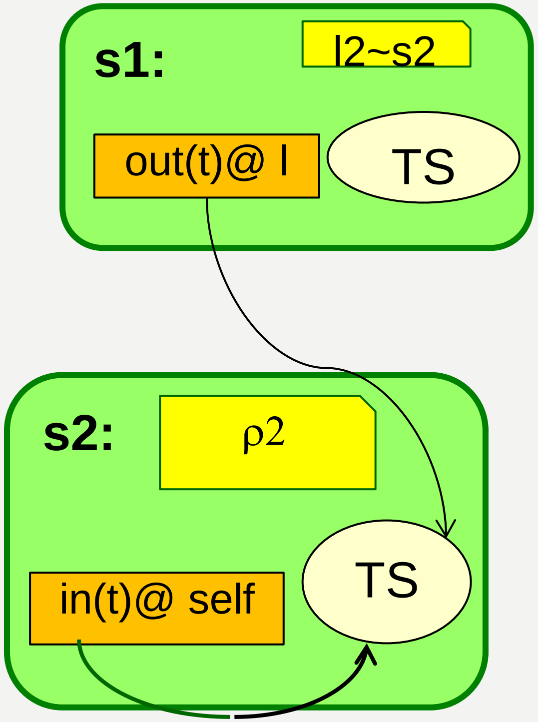


Example

$$s_1 ::_{[s_1/\mathrm{self}]\bullet[s_2/l_2]} out(1)@l_2.nil \parallel s_2 ::_{[s_2/\mathrm{self}]} in(1)@self.nil$$

Syntax

$$\begin{array}{rll}
\text{Nets:} & N ::= & 0 \mid s ::_\rho C \mid N_1 \parallel N_2 \mid (\nu s)N \\
\text{Components:} & C ::= & \langle t \rangle \mid P \mid C_1 | C_2 \\
\text{Processes:} & P ::= & \mathbf{nil} \mid a.P \mid P_1 | P_2 \mid X \mid \mathbf{rec}\, X.P \\
\text{Actions:} & a ::= & \mathbf{in}(T)@u \mid \mathbf{out}(t)@u \mid \mathbf{new}(s) \\
\text{Tuples:} & t ::= & u \mid P \mid t_1, t_2 \\
\text{Templates:} & T ::= & u \mid !x \mid !X \mid T_1, T_2
\end{array}$$



Structured Reduction semantics

- describes the process behavior in a net

$$(\textsc{Red-Out})\ \frac{\rho(u) = s' \qquad \mathcal{E}[\![t]\!]_\rho = t'}{s ::_\rho \mathbf{out}(t)@u.P \parallel s' ::_{\rho'} \mathbf{nil} \ \longmapsto\ s ::_\rho P \parallel s' ::_{\rho'} \langle t' \rangle}$$

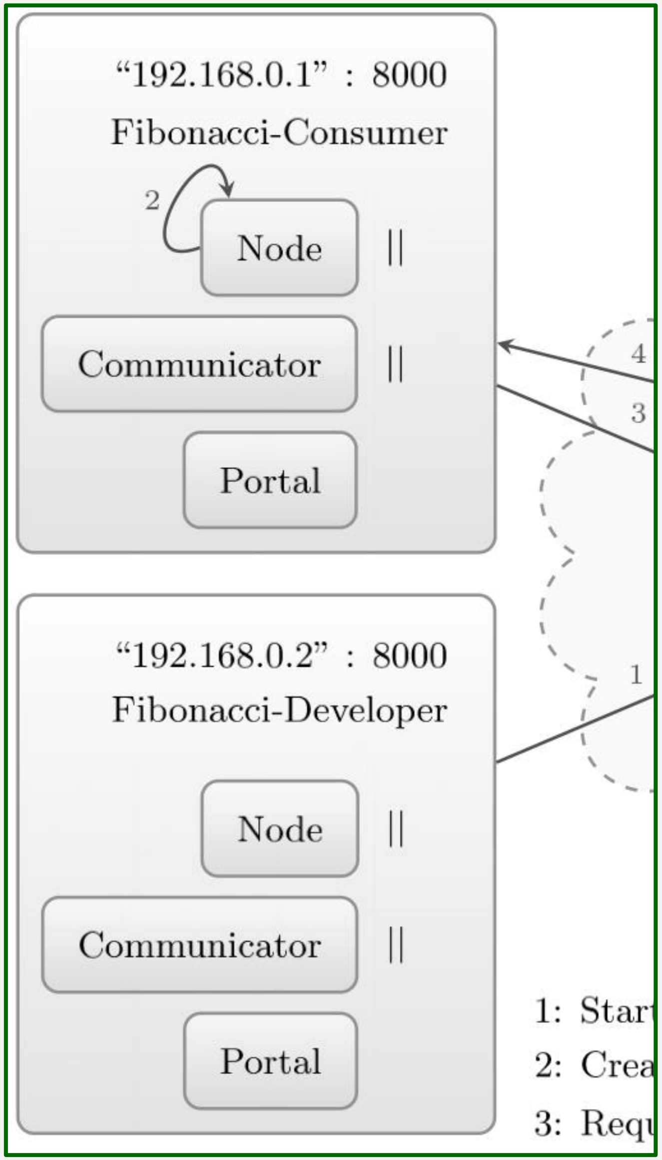
We developed three Maude-based implementations of KLAIM:

- **M-KLAIM**

  a formal executable specification of KLAIM
- **MP-KLAIM**

  an refinement of **M-KLAIM** for asynchronous message-passing specification
- **D-KLAIM**

  an extension of **MP-KLAIM** for distributed execution (communication through sockets)



Questions

We used the *-KLAIM implementations for simulation and analysis with the Maude tools such as

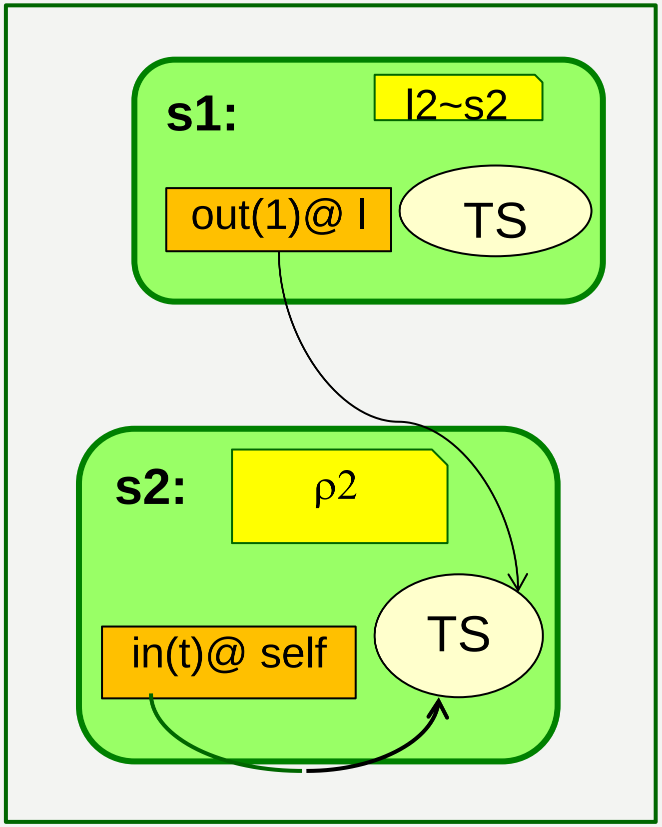
- Distributing a cloud service over a several Maude runtimes
- LTL-model checking of a mutual exclusion algorithm
- State space analysis of a load balancer using the Maude search command

but

- **What are the semantic relationships of** *-KLAIM **with KLAIM?**
- **Which properties are preserved?**

M-KLAIM syntax

- Direct correspondence to KLAIM syntax



KLAIM:

$$s_1 ::_{[s_1/\mathrm{self}]\bullet[s_2/l_2]} out(1)@l_2.nil \parallel s_2 ::_{[s_2/\mathrm{self}]} in(1)@self.nil$$

M-KLAIM:

```
(site('1)::{[site('1)/self]*[site('2)/'l2]} out(1)@'l2.nil) ||
(site('2)::{[site('2)/self]} in(1)@self.nil)
```

Rewriting semantics of KLAIM:

- Reduction semantics can be naturally expressed in rewriting logic

KLAIM:

$$(\textsc{Red-Out})\ \frac{\rho(u) = s' \qquad \mathcal{E}[\![t]\!]_\rho = t'}{s ::_\rho \mathbf{out}(t)@u.P \parallel s' ::_{\rho'} \mathbf{nil} \ \longmapsto\ s ::_\rho P \parallel s' ::_{\rho'} \langle t' \rangle}$$

M-KLAIM:

```
crl [out-remote] :
   (S1::{RHO1} (out(T) @ L) . SP | PP) || (S2::{RHO2} P)
=>
   (S1::{RHO1} SP | PP) || (S2::{RHO2} P | <T[| T |]RHO> )
if S2 := RHO1(L) .
```

Definitions and Notations

- (A, –>) (Unlabelled) transition system
- (A, –>, L) Kripke structure where
    - L: A –> $\mathcal{P}$(AP) labeling function,
    - AP set of atomic propositions

- Logic CTL*(AP)

$$
\begin{array}{ll}
\text{state formulas:} & \varphi ::= p \in AP \mid \top \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{A}\psi \mid \mathbf{E}\psi \\
\text{path formulas:} & \psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi \mid \psi\mathbf{R}\psi \mid \mathbf{G}\psi \mid \mathbf{F}\psi\,.
\end{array}
$$

- ACTL*(AP): CTL* formulas in negation normal form
- ACTL*(AP) \ X: ACTL* formulas without next-operator
- ACTL*(AP) \ "not": ACTL* formulas without negation

Simulation

- **Simulation of tss** (A, $\rightarrow_A$) by (B, $\rightarrow_B$) is a binary relation ~> s.th.

  if a ~> b and a $\rightarrow_A$ a′ then there is b′ with b $\rightarrow_B$ b′ and a′ ~> b′
- **AP-simulation** of (A, $\rightarrow_A$, L_A) by (A, $\rightarrow_B$, L_B) is a simulation of tss

  s.th. if a ~> b then L_B(b) is a subset of L_A(a).
  - ~> is **strict** if a ~> b implies $L_B(b) = L_A(a)$
- **Bisimulation, AP-bisimul**ation: as usual.
- ~> **reflects the satisfaction of a formula** ϕ if B,b $\models \phi$ and a ~> b imply A,a $\models \phi$

- **Theorem (Clarke, Grumberg, Peled 1999)**

  AP-simulations reflect the satisfaction of ACTL*(AP) \ not(AP) formulas, strict simulations reflect the satisfaction of ACTL*(AP) formulas.

KLAIM and M-KLAIM are Bisimilar

- The KLAIM and M-KLAIM semantics are transition systems:

$$TS_{\text{KLAIM}} = (\text{KLAIM-NET}, \longmapsto)$$

$$TS_{\text{M-KLAIM}} = (\mathcal{T}(\text{M-KLAIM})_{net}, \Rightarrow_1)$$

where KLAIM-Net denotes all ground KLAIM terms of sort net,

$\mathcal{T}$(M-KLAIM)$_{net}$ all ground valid M-KLAIM terms of sort Net and $=>_1$ the one-step rewrite relation.

- Theorem 1

TS_{KLAIM} and $TS_{M\text{-}KLAIM}$ are bisimilar w.r.t.

$N \sim M$ if $N \equiv \nu s_1 \ldots \nu s_k.\ m2k(M)$

where m2k translates M-KLAIM terms into KLAIM:

$$\begin{aligned} m2k(s\,\{c\} :: \{\rho\}\, p) &= s ::_{m2k(\rho)} m2k_{s,c}(p) \\ m2k(n_1 \,||\, n_2) &= m2k(n_1) \,||\, m2k(n_2) \\ m2k_{s,c}(newloc(lvn).p) &= new(g(s,c)).m2k_{s,c}(p) \end{aligned}$$

KLAIM and M-KLAIM are Bisimilar

- Extend TS_{KLAIM} and $TS_{M\text{-}KLAIM}$ to Kripke structures by choosing

  AP to be a subset of $\{p_t \mid t \text{ ground KLAIM term}\}$

  such that

  $N \models p_t$ iff $M \models p_t$ and $N \equiv \nu s_1 \ldots \nu s_k.\ m2k(M)$

- Example: $M \models p_{s,t}$ iff $N = \nu s_1 \ldots \nu s_k.\ (s ::_\rho \langle t \rangle \mid P) \parallel R$

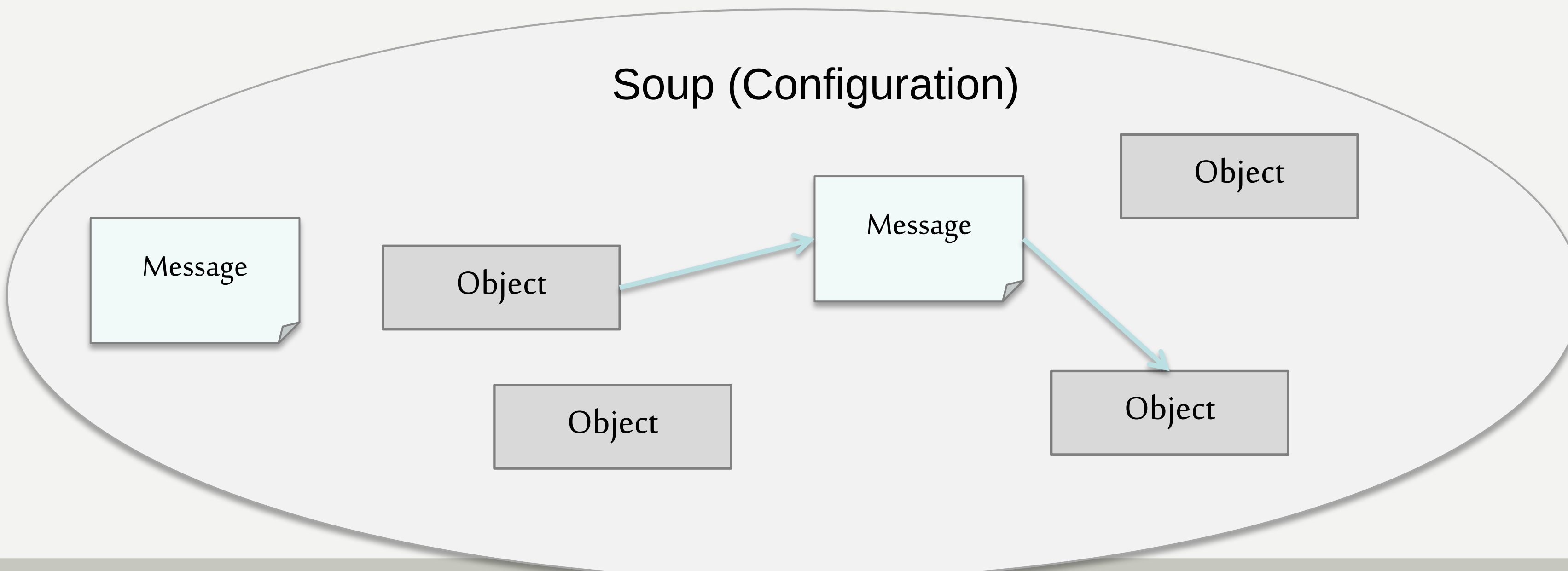
- Corollary 1

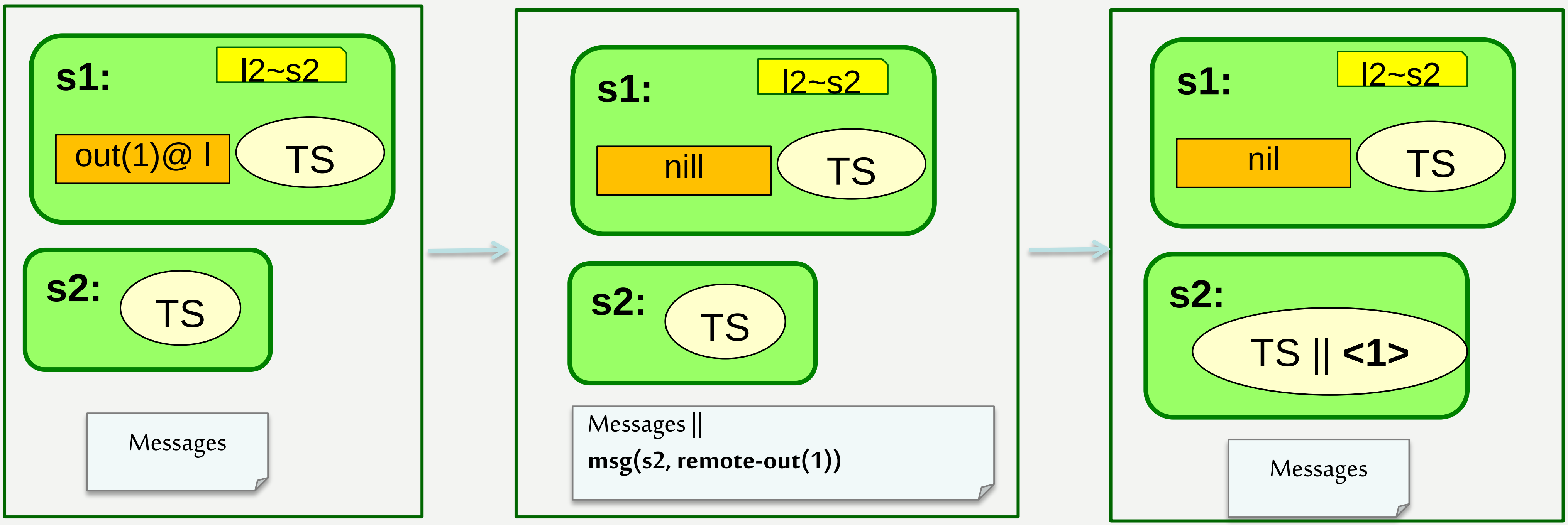
  TS_{KLAIM} and $TS_{M\text{-}KLAIM}$ are AP-bisimilar and

  reflect the satisfaction of ACTL*(AP) formulas.

Maude supports modeling of distributed **object-based systems** in which objects communicate asynchronously via message passing

- Message passing is a natural way of expressing communication in distributed systems
- We alter the KLAIM semantics by introducing asynchronous inter-node communication





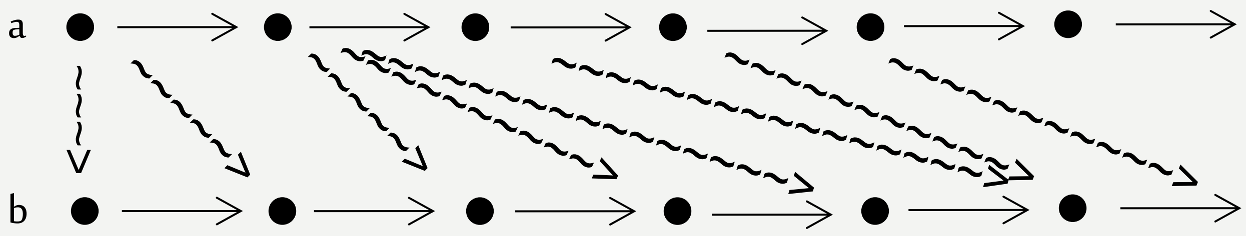
- An out-action is split into two steps:
  - Producing an out-message an sending it into the “soup”
  - Consuming the out-message by inserting the contents into the tuple space

Out-rules formally:

```
crl [out-remote-produce] :
   (S1::{RHO} (out(T) @ L) . SP | PP)
=>
   (S1::{RHO} SP | PP) || msg(S2, remote-out(T[| T |]RHO))
if S2 := RHO(L) /\ S2 =/= S1 .
```

```
rl [out-remote-consume] :
   (S::{RHO} PP) || msg(S, remote-out(ET))
=>
   (S::{RHO} PP | <ET>) .
```

- **Matching path**



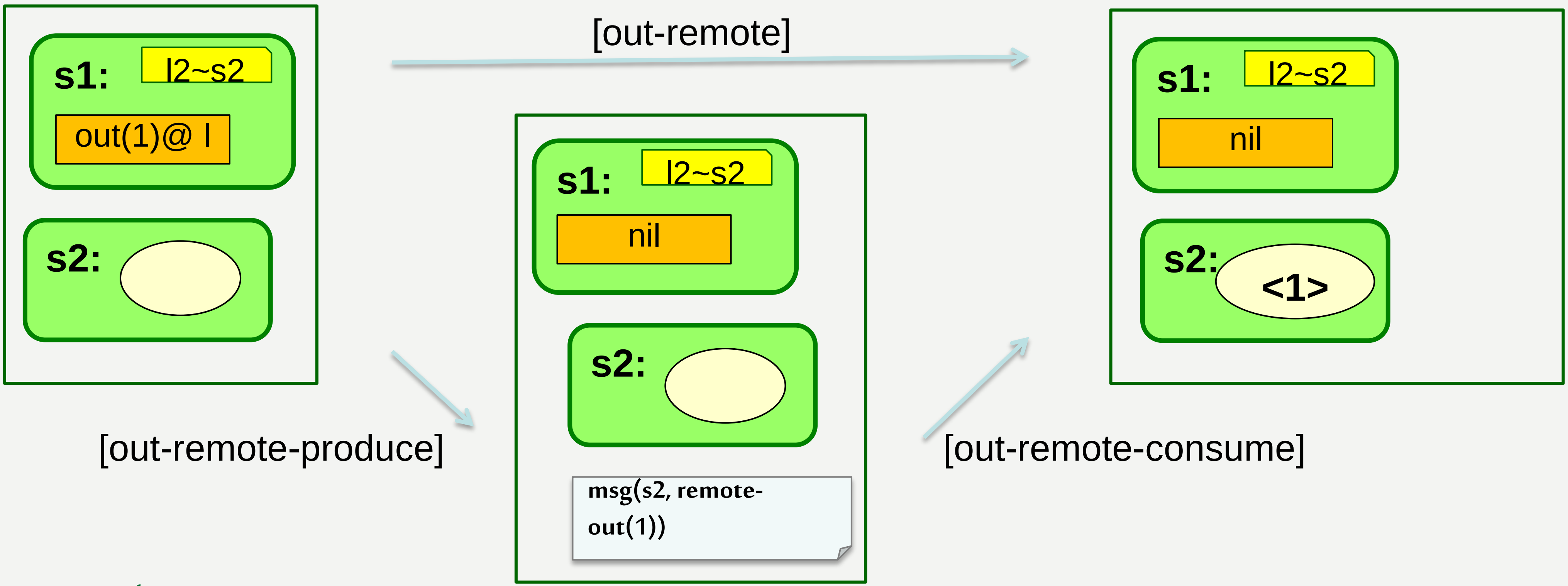
- **Stuttering Simulation of tss** (A, $\rightarrow_A$) by (B, $\rightarrow_B$) is a binary relation ~> s.th.
  if for each a ~> b and each path starting at a there is a matching path starting at b
- **Stuttering AP-Simulation** as before

- **Theorem (Meseguer, Palomino, Marti-Oliet 2010)**

  Stuttering AP-simulations reflect the satisfaction of ACTL*(AP) \ (X, not)(AP) formulas;

  strict simulations reflect the satisfaction of ACTL*(AP) \ X(AP) formulas

MP-KLAIM is a Stuttering Simulation of M-KLAIM



- Theorem 2

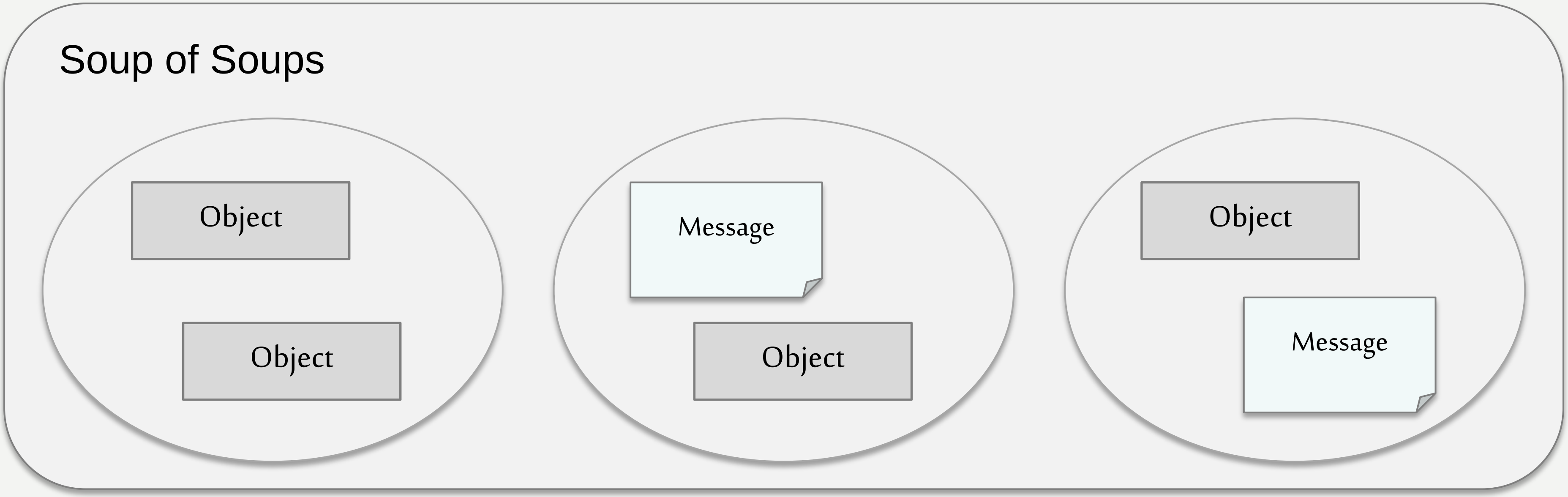
  $TS_{MP\text{-}KLAIM}$ is a strict stuttering AP-Simulation of $TS_{M\text{-}KLAIM}$ and thus reflects the satisfaction of ACTL*(AP) \ X(AP) formulas.

- But satisfaction of atomic props is often nonstandard:

$$M \models p_{s,t} \quad \text{iff} \quad M = (s ::\rho <t> \mid P) \parallel R \text{ or } M = (s ::\rho\ P) \parallel <t> \parallel R$$

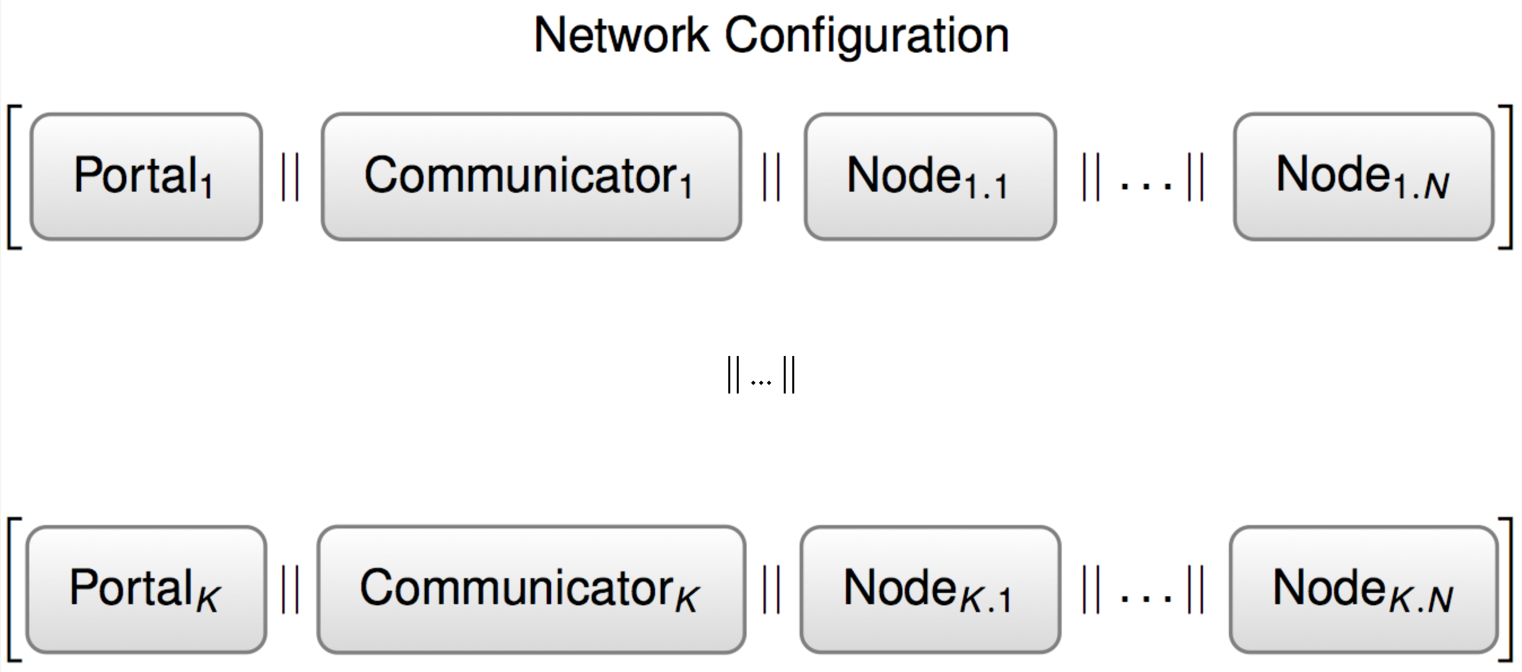
The **D-KLAIM** extension allows multiple instances of Maude to execute specifications based on MP-KLAIM.

- Instances communicate through sockets
- Socket communication is supported by rewriting with external objects in Maude
- D-KLAIM introduces objects to handle the socket communication
- D-KLAIM uses a buffered approach for reliable communication



D-KLAIM Socket Abstraction

- For formal analysis we developed a **socket abstraction** that captures the **behavior of Maude's socket capabilities** inside a Maude specification.



D-KLAIM Socket Abstraction

- The communicator wraps a message addressed to another instance in a transfer message:

```
crl [transfer-send] :
  (communicator(IP1 : PORT1) :: ATTS)
    || msg(IP2 : PORT2 # ID, MSG)
=>
  (communicator(IP1 : PORT) :: ATTS)
    || transfer-send(msg(IP2 : PORT2 # ID, MSG))
if otherPhysicalNode(IP1, PORT1, IP2, PORT2) .
```

- The comm

```
rl [portal-transfer] :
  [ < portal :: ip : IP1, port : PORT1 >
    || transfer-send(msg(IP2 : PORT2 # ID, MSG)) || N1 ]
  || [ < portal :: ip : IP2, port : PORT2 > || N2 ]
=>
  [ < portal :: ip : IP1, port : PORT1 > || N1 ]
  || [ < portal :: ip : IP2, port : PORT2 >
    || transfer-receive(msg(IP2 : PORT2 # ID, MSG)) || N2 ] .
```

- Then it is

D-KLAIM and MP-KLAIM are Stuttering Bisimilar

- Stuttering bisimulation:
  - Self-actions are the same in MP-KLAIM and D-KLAIM
  - Transfer-actions are stuttering actions and complement the actions communicating with nodes of another site.

- Theorem 3

$TS_{MP\text{-}KLAIM}$ and $TS_{D\text{-}KLAIM}$ are stuttering bisimilar and thus reflect the satisfaction of ACTL*(AP) \ X(AP) formulas.

Concluding Remarks

- *-KLAIM provides provably correct implementations of KLAIM
  - Related with KLAIM by

    bisimulation, stuttering simulation and

    stuttering bisimulation
  - .Reflecting ACTL*(AP) formulas
- Future Work
  - Transition to full socket specification
  - Strengthening the transition to MP-KLAIM
    - Fairness assumptions
    - Real-time architectural patterns (PALS)
  - Analyzing novel formalisms such as SCEL