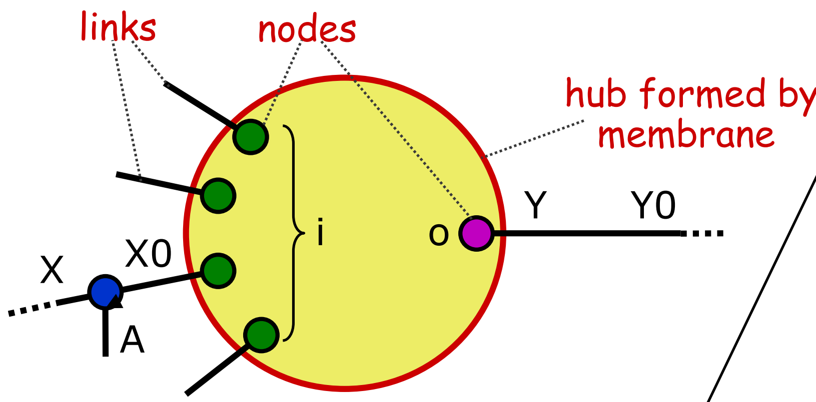
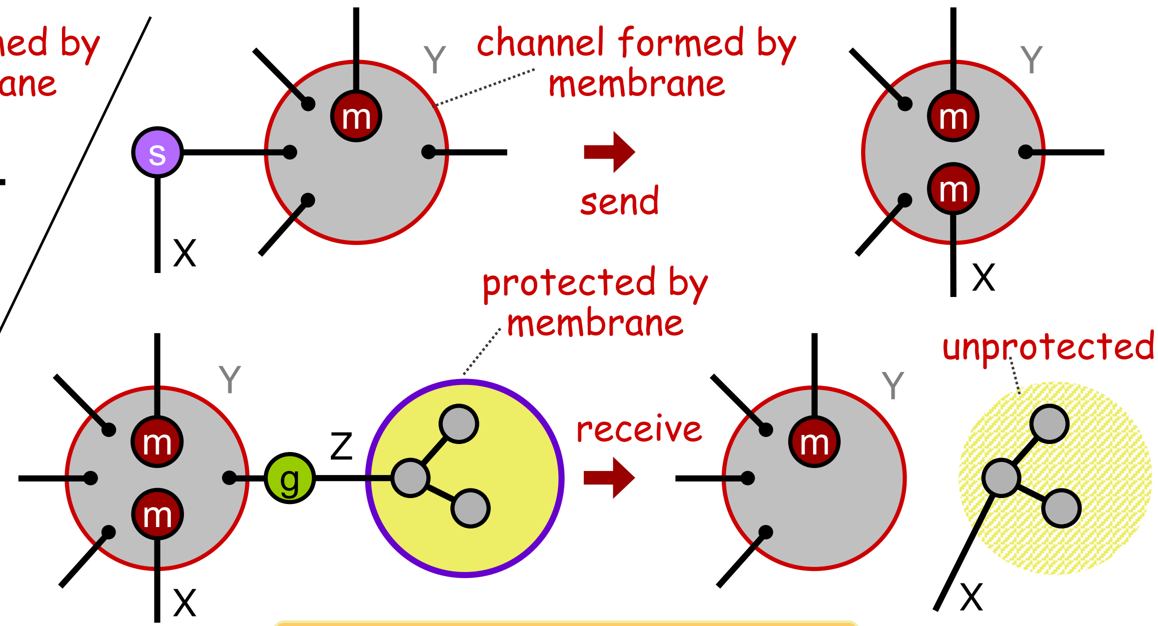
Encoding Distributed Process Calculi into LMNtal

Kazunori Ueda
Dept. of Computer Science, Waseda University
September 2006

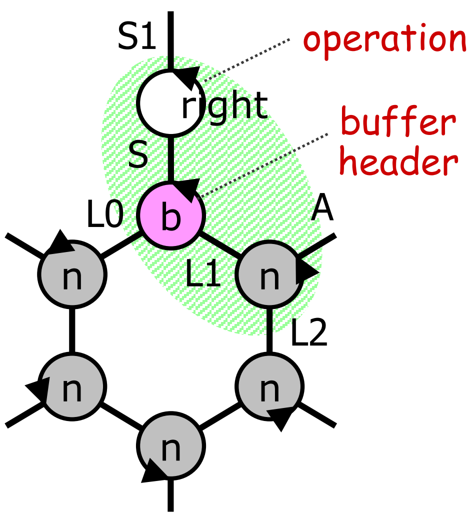
LMNtal allows us to represent computation in terms of hierarchical graph rewriting



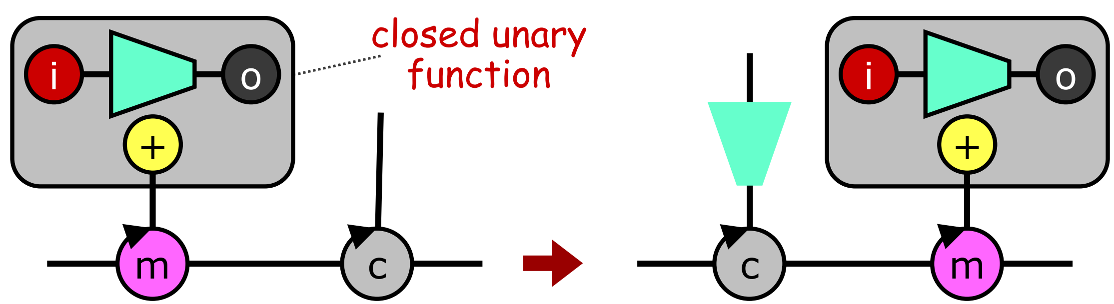
n-to-1 comm.



asynchronous π-calculus



cyclic structures



map function

LMNtal in a Nutshell

- Rule-based concurrent **language** for expressing & rewriting both **connectivity** and **hierarchy**
- Substrate **model** of X-calculi (X = lambda, pi, ambient, . . .), multiset rewriting, etc.
- Computation is manipulation of **diagrams**
  - **Links** express 1-to-1 **connectivity**
  - **Membranes** express **hierarchy** and **locality** of rules and data
  - Allows **programming by self-organization**
  - Good also for **knowledge representation**

Syntax and Semantics, in one slide

$$\begin{array}{rll} \text{(Process)} & P ::= \mathbf{0} \mid p(X_1,\ldots,X_m) \mid P,P \mid \{P\} \mid T :\!-\, T \\ \text{(Process template)} & T ::= \mathbf{0} \mid p(X_1,\ldots,X_m) \mid T,T \mid \{T\} \mid T :\!-\, T \\ & \quad \mid \texttt{@}p \mid \texttt{\$}p[X_1,\ldots,X_m \,|\, A] \mid p(*X_1,\ldots,*X_n) \\ \text{(Residual)} & A ::= \texttt{[]} \mid *X \end{array}$$

$$\text{(E1)}\quad \mathbf{0},P \equiv P \qquad \text{(E2)}\quad P,Q \equiv Q,P \qquad \text{(E3)}\quad P,(Q,R) \equiv (P,Q),R$$

$$\text{(E4)}\quad P \equiv P[Y/X] \qquad \text{if } X \text{ is a local link of } P$$

$$\text{(E5)}\quad P \equiv P' \;\Rightarrow\; P,Q \equiv P',Q \qquad \text{(E6)}\quad P \equiv P' \;\Rightarrow\; \{P\} \equiv \{P'\}$$

$$\text{(E7)}\quad X = X \equiv \mathbf{0} \qquad \text{(E8)}\quad X = Y \equiv Y = X$$

$$\text{(E9)}\quad X = Y,\ P \equiv P[Y/X] \qquad \text{if } P \text{ is an atom and } X \text{ occurs free in } P$$

$$\text{(E10)}\quad \{X = Y,\ P\} \equiv X = Y,\ \{P\} \qquad \text{if exactly one of } X \text{ and } Y \text{ occurs free in } P$$

$$\text{(R1)}\ \frac{P \longrightarrow P'}{P,Q \longrightarrow P',Q} \qquad \text{(R2)}\ \frac{P \longrightarrow P'}{\{P\} \longrightarrow \{P'\}} \qquad \text{(R3)}\ \frac{Q \equiv P \quad P \longrightarrow P' \quad P' \equiv Q'}{Q \longrightarrow Q'}$$

$$\text{(R4)}\quad \{X = Y, P\} \longrightarrow X = Y,\ \{P\} \qquad \text{if } X \text{ and } Y \text{ occur free in } \{X = Y, P\}$$

$$\text{(R5)}\quad X = Y,\ \{P\} \longrightarrow \{X = Y, P\} \qquad \text{if } X \text{ and } Y \text{ occur free in } P$$

$$\text{(R6)}\quad T\theta, (T :\!-\, U) \longrightarrow U\theta, (T :\!-\, U)$$

Towards a Unifying Rule-based Language

- The calculi encoded in LMNtal include:
  - λ-calculus ([new] nondeterministic, call-by-name) based on graph reduction
    - β-reduction, δ-reduction, graph copying
  - π-calculus ([new] synchronous, asynchronous)
    - names as cells
  - [new] Ambient calculus (this talk)
  - CHR (Constraint Handling Rules)
  - (more calculi underway)

Distributed Process Calculi

- Generic name for process calculi with the notion of locations and locality
- Membranes are typically used for representing (delimiting) locations
- Ambient Calculus is the best studied formalism, with many similarities with LMNtal
  - hierarchical membranes
  - reconfiguration (mobility)
  - no remote actions

Ambients

- A bounded place where computation happens.
  - The place is delimited by explicit boundaries (cf. membranes)
- An ambient can contain other ambients
- An ambient can migrate across boundaries of other ambients (if agreed upon between the both camps)
- Interprocess communication respects ambient boundaries

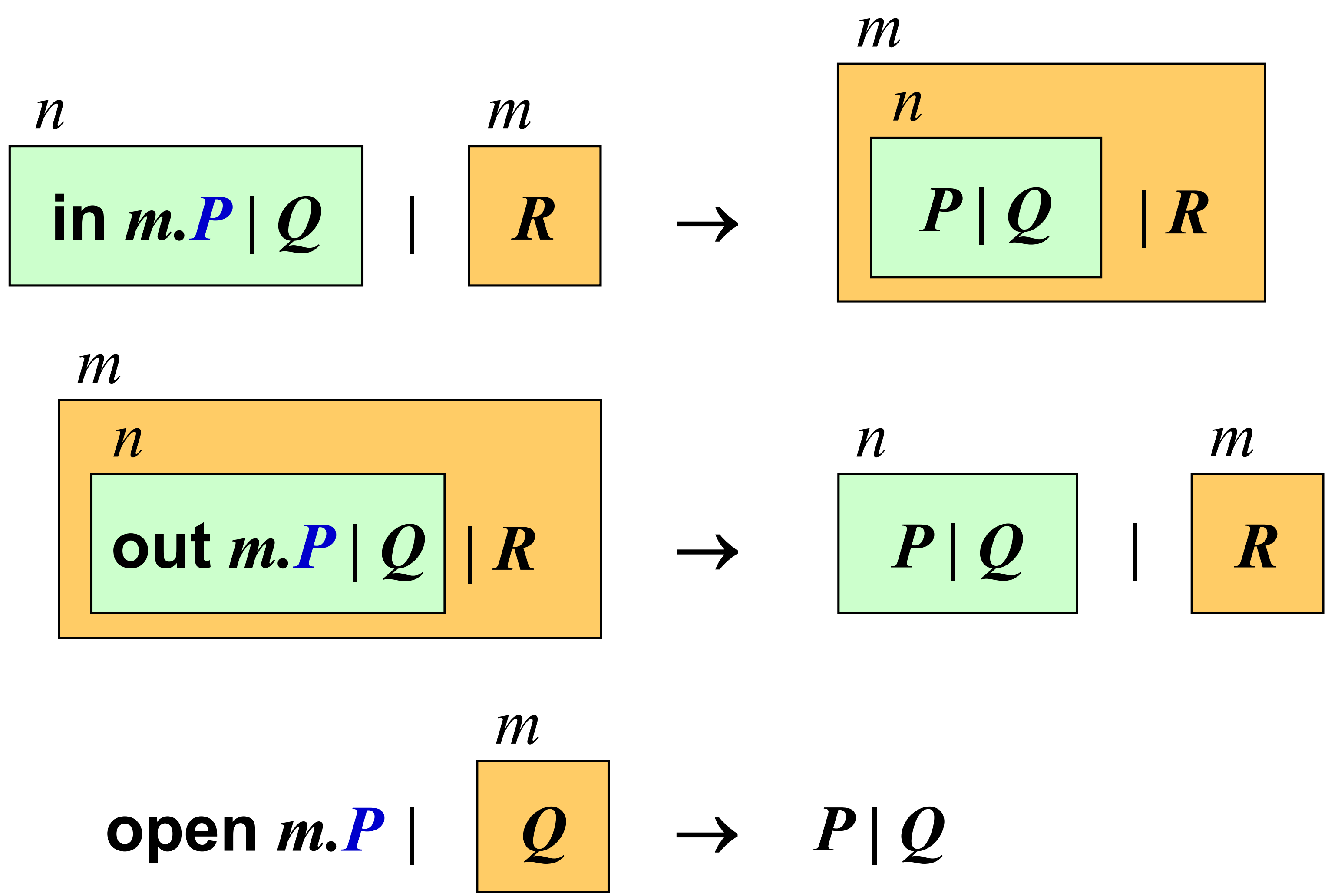
Structure of an Ambient

- Each ambient has its own name used for access control (enter / exit / communication)
- Each ambient has its own collection of agents (processes) executed inside the ambient.
- The top-level agent of an ambient takes care of migration

Ambient Calculus (Pure Mobility Calculus)

(names)	n			
(processes)	P, Q	$::=$	$(\nu n)P$	(restriction)
			$\mathbf{0}$	(inactivity)
			$P \mid Q$	(composition)
			$!P$	(replication)
			$n[P]$	(ambient)
			$M.P$	(action)
(capabilities)	M	$::=$	$\mathsf{in}\ n$	(can enter n)
			$\mathsf{out}\ n$	(can exit n)
			$\mathsf{open}\ n$	(can open n)

Actions



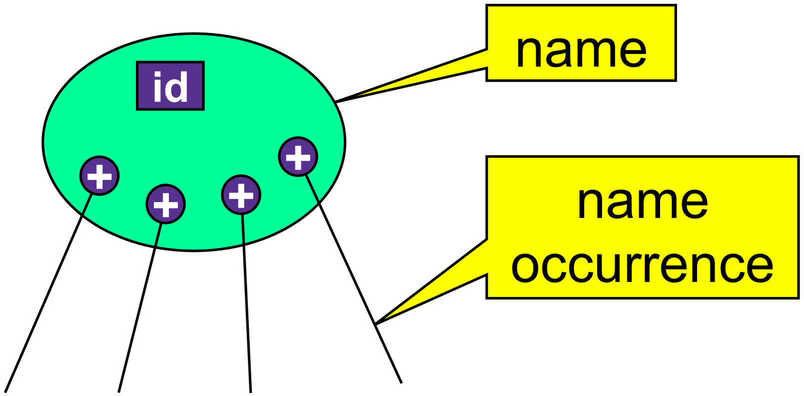
Ambient Names

- As in the π-calculus, names play important roles in the ambient calculus
- Basic operations, interrelated to each other
  - (a) create a fresh local name (secret keys)
  - (b) pass it around
  - (c) name an ambient
  - (d) associate with capabilities (= mobility operations)
- The main issue in encoding into LMNtal is how to represent names.

Encoding of Names, Two Alternatives

(Ambient Calculus)		(LMNtal)
Ambient names	→	atom names
Ambient names	→	hierarchical graphs

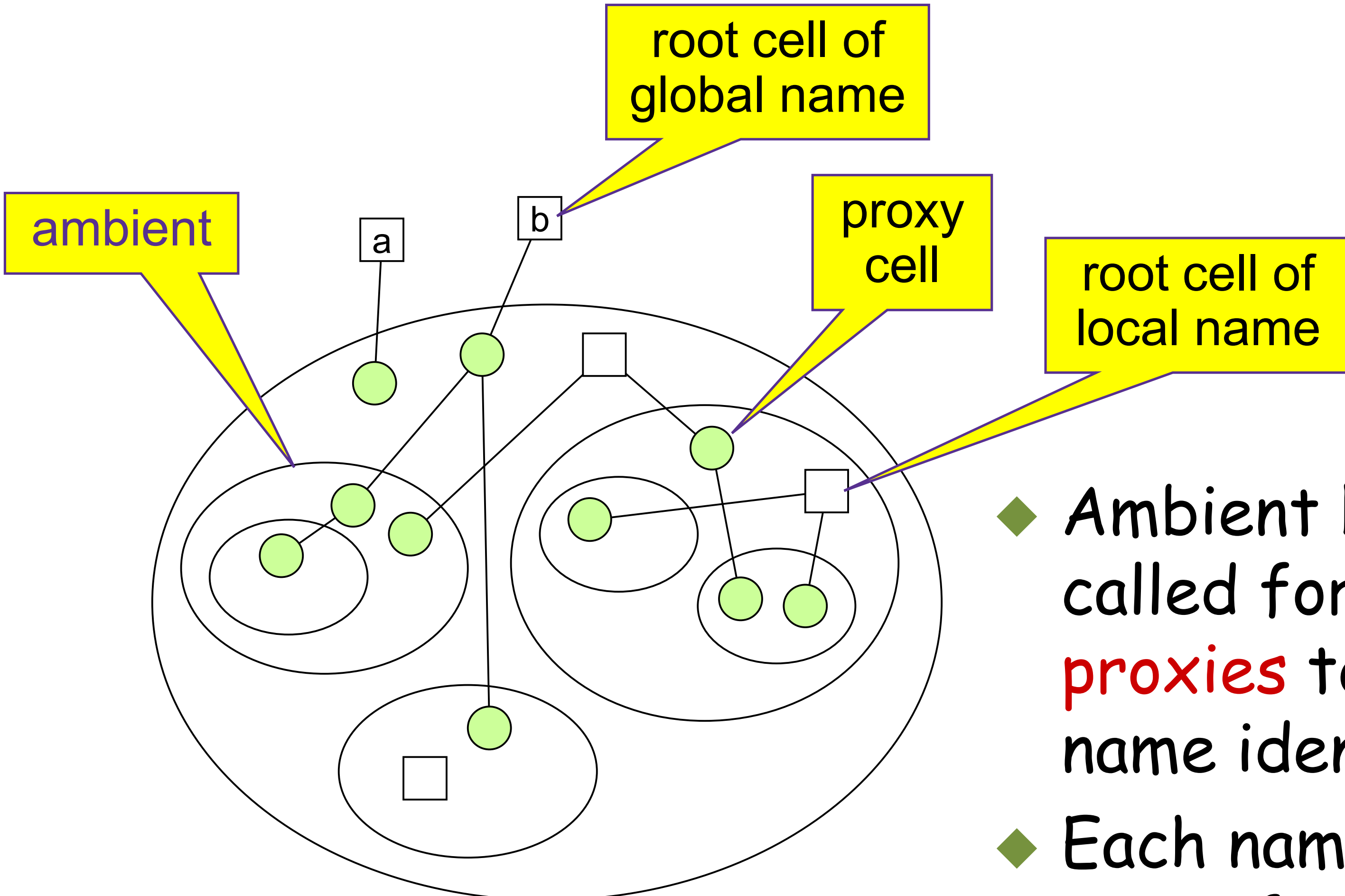
- We choose the latter
  - to make reference structures explicit
  - to handle local names
  - to use atom names to encode fixed language constructs (in/out/open) only



Encoding Ambients in LMNtal

(Ambient Calculus)	(LMNtal)
global name	cell {id, name(n), +N_1, ..., +N_k}
name reference	incident link N_i
local name (**νn**)	cell without name(n)
name proxy	cell {id, +N_1, ..., +N_k, -N }
ambient	cell {a.use, amb(n), ... }
// composition $P \mid Q$	multiset
action $M.P$	in/2, out/2, open/2
action body P	process enclosed by membrane

Name Trees



- Ambient hierarchies called for name proxies to recognize name identity locally
- Each name forms a tree of name proxies
- Normal form of a name tree should correspond to an ambient hierarchy

Encoding Ambients, Formally

$$[\![\mathbf{0}]\!] \stackrel{\text{def}}{=} \mathbf{0}$$

$$[\![P \mid Q]\!] \stackrel{\text{def}}{=} ([\![P]\!]\texttt{,}[\![Q]\!])\!\downarrow$$

normalization of name trees

$$[\![(\nu n)P]\!] \stackrel{\text{def}}{=} (\text{hide}_n([\![P]\!]\!\downarrow))\!\downarrow$$

hide the name n

$$[\![n[P]]\!] \stackrel{\text{def}}{=} \texttt{\{}@amb\texttt{,amb(L),}[\![n]\!]\texttt{(L),}[\![P]\!]\texttt{\}}\!\downarrow$$

$$[\![M.P]\!] \stackrel{\text{def}}{=} ([\![M]\!]([\![P]\!]))\!\downarrow$$

$$[\![op\ n]\!] \stackrel{\text{def}}{=} [\![op]\!]([\![n]\!]) \quad (op \in \{\mathsf{in}, \mathsf{out}, \mathsf{open}\})$$

$$[\![op]\!] \stackrel{\text{def}}{=} \lambda f.\lambda p.(op\texttt{(L,M),\{+M,}p\texttt{\},}f\texttt{(L)})$$

$$(op \in \{\mathsf{in}, \mathsf{out}, \mathsf{open}\})$$

$$[\![n]\!] \stackrel{\text{def}}{=} \lambda l.\texttt{\{id,name(}n\texttt{),+}l\texttt{\}}$$

Encoding Actions

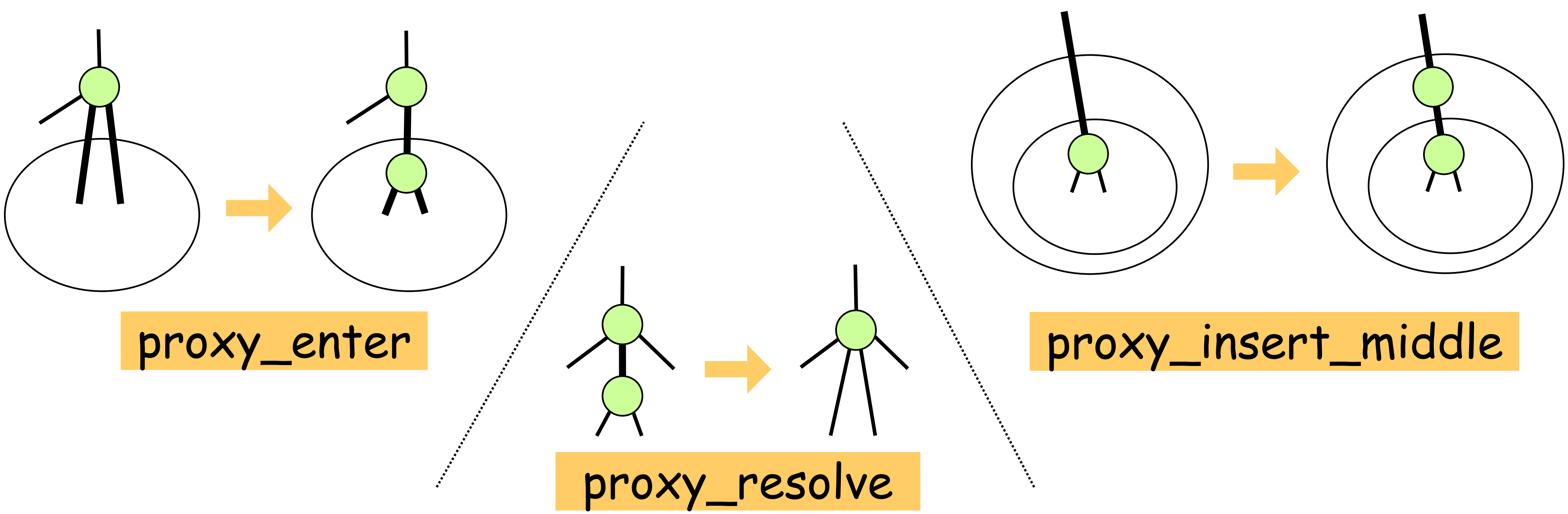
- LMNtal rules for in/out/open are the literal translation of the original operational semantics.

```
/* n[in m.P | Q] | m[R] --> m[n[P|Q] | R] */
in@@
{amb(N0), {id,+N0,$n}, {id,+M0,-M1,$m0}, in(M0,{$p}), $q,@q},
{amb(M2), {id,+M2,-M3,$m1}, $r,@r},
{id,+M1,+M3,$m2} :-
   {amb(M4), {id,+M4,+M5,-M,$m1},
      {amb(N2), {id,+N2,$n}, {id,-M5,$m0}, $p,$q,@q},
   $r,@r},
   {id,+M,$m2}.
```

. . . (similarly for out and open) . . .

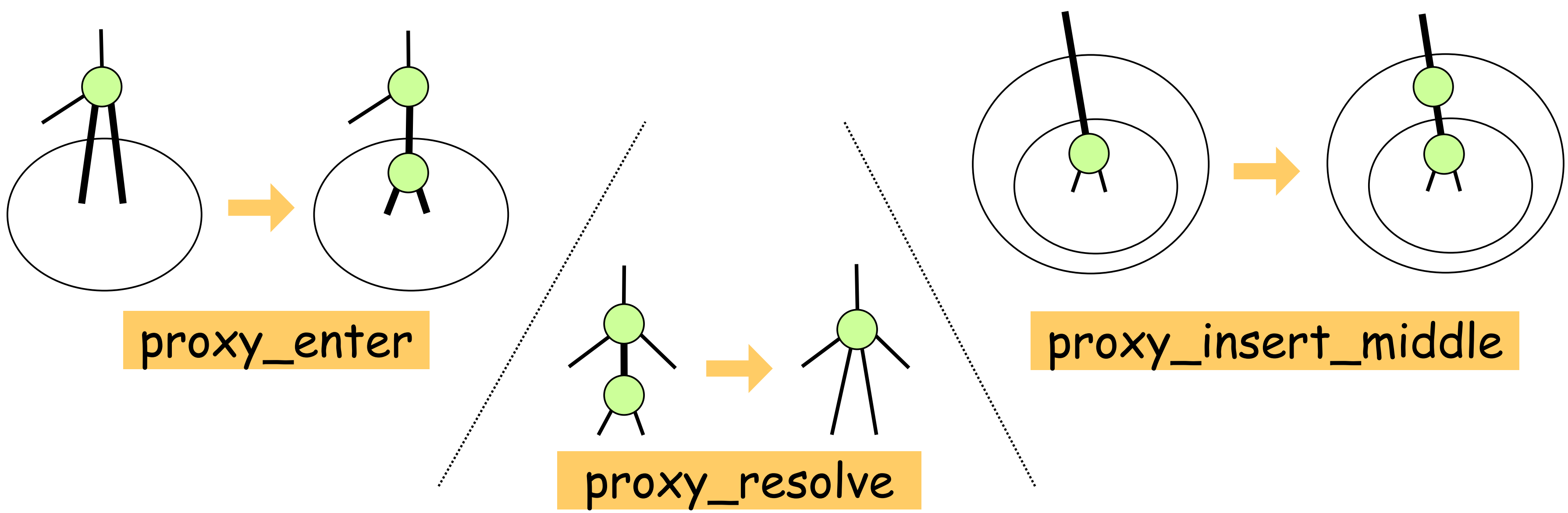
Name Tree Normalization

- in/out/open moves indefinite number of name references across ambient boundaries, violating the normal form conditions temporarily
- Name trees are reformed autonomously and asynchronously
- Examples:



Name Tree Normalization

- [invariant] Both in/out/open actions and asynchronous reformation preserve connectivity (of names cells representing a name).
- [partial correctness] A name tree is in (unique) normal form iff no reformation rules apply.
- [total correctness] Exercise.



Examples and Demonstration

1. Locks
2. Mobile agent authentication
3. Firewall access
4. Objective moves
5. Choice
6. Rer

$$\begin{aligned} \mathit{Firewall} &\stackrel{\text{def}}{=} (\nu w) w[k[\mathsf{out}\ w\,.\,\mathsf{in}\ kk\,.\,\mathsf{in}\ w] \\ &\qquad \mid \mathsf{open}\ kk\,.\,\mathsf{open}\ kkk\,.\,P] \\ \mathit{Agent} &\stackrel{\text{def}}{=} kk[\mathsf{open}\ k\,.\,kkk[Q]] \end{aligned}$$

Uses of LMNtal Membranes

- Encoding of the ambient calculus makes heavy use of membranes
  - names and name proxies (no rulesets)
  - ambients (with rulesets)
  - action body (for protection)
- Type systems should be able to infer different uses
- Planned: lightweight/featherweight membranes

Encoding Replication

- Replication is defined in terms of structural congruence: $!P \equiv P \mid !P$
  - Use of ! in the AC: to encode procedures
    - $!(\text{open } n \,.\, Q) \mid n[\,] \rightarrow !(\text{open } n \,.\, Q) \mid Q$
  - P should be spawned on demand
    - otherwise it causes divergence
  - Current solution: to encode $!(\text{open } n \,.\, P) \mid n[Q] \rightarrow !(\text{open } n \,.\, P) \mid P \mid Q$
  - Copying P may increase references to names (i.e., name tree leafs) indefinitely
    - handled by aggregates or nlmem API

Conclusions

- Migration of ambients involves migration of name (= resource) accesses across administrative domains. Our encoding of the AC into LMNtal has
  - made the topology of name accesses explicit and
  - given an autonomous and asynchronous algorithm for name tree management.
- The encoding consists of
  - 3 rules for the basic operations,
  - 8 rules for name tree management, and
  - 4 rules for GC,

  all allowing graphical interpretation.