Cours de DEA

# Informal introduction to Mobility

Jérôme Feret
École normale supérieure
http://www.di.ens.fr/~feret

February, 2004

# Overview

1. Mobile systems:
   (a) Definition,
   (b) Examples,
   (c) A model for mobile systems;
2. Other forms of mobility:
   (a) Mobilty via migration,
   (b) "Restricted" mobility;
3. Shared-memory modelisation:
   (a) Naive description,
   (b) Synchronization,
   (c) Mutex.

# Mobile systems

# Mobile system

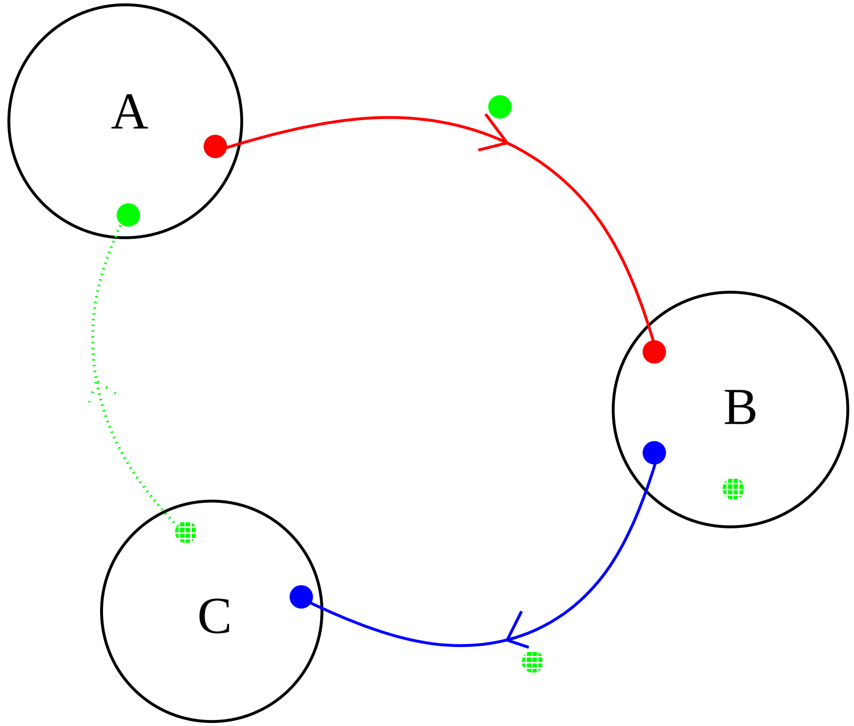
A pool of processes which interact and communicate:

Interactions:

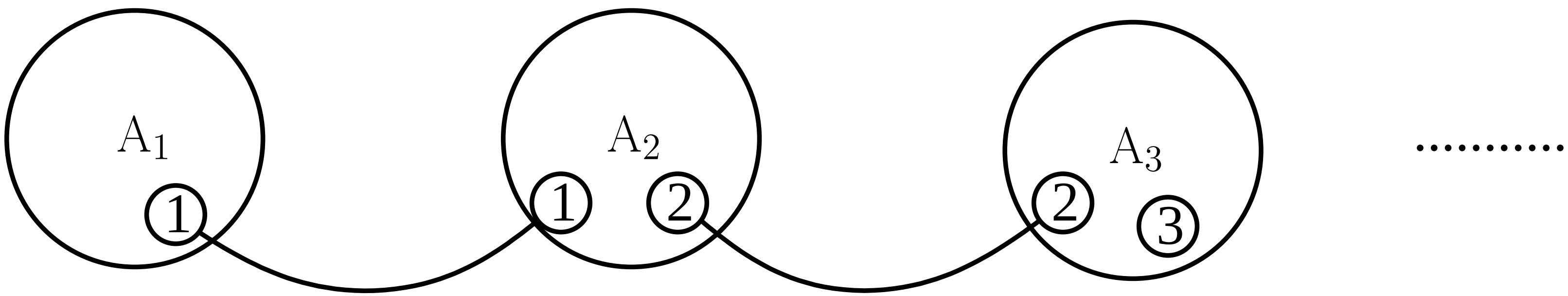
- synchronize process computation;
- change process structure (communication, migration);
- change communication links;
- create new processes.

**Topology of interaction may be unbounded !**

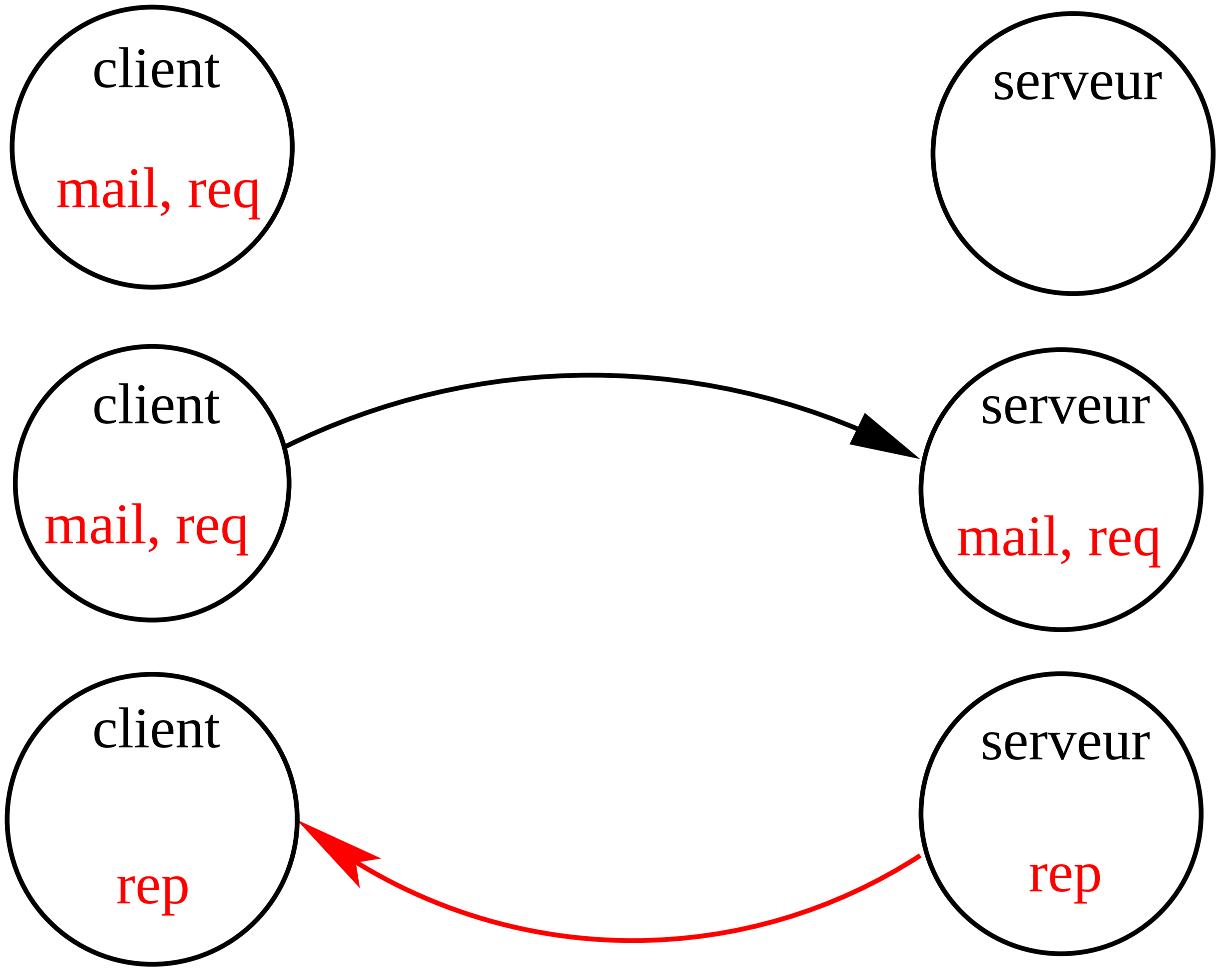
# Dynamic linkage of agents



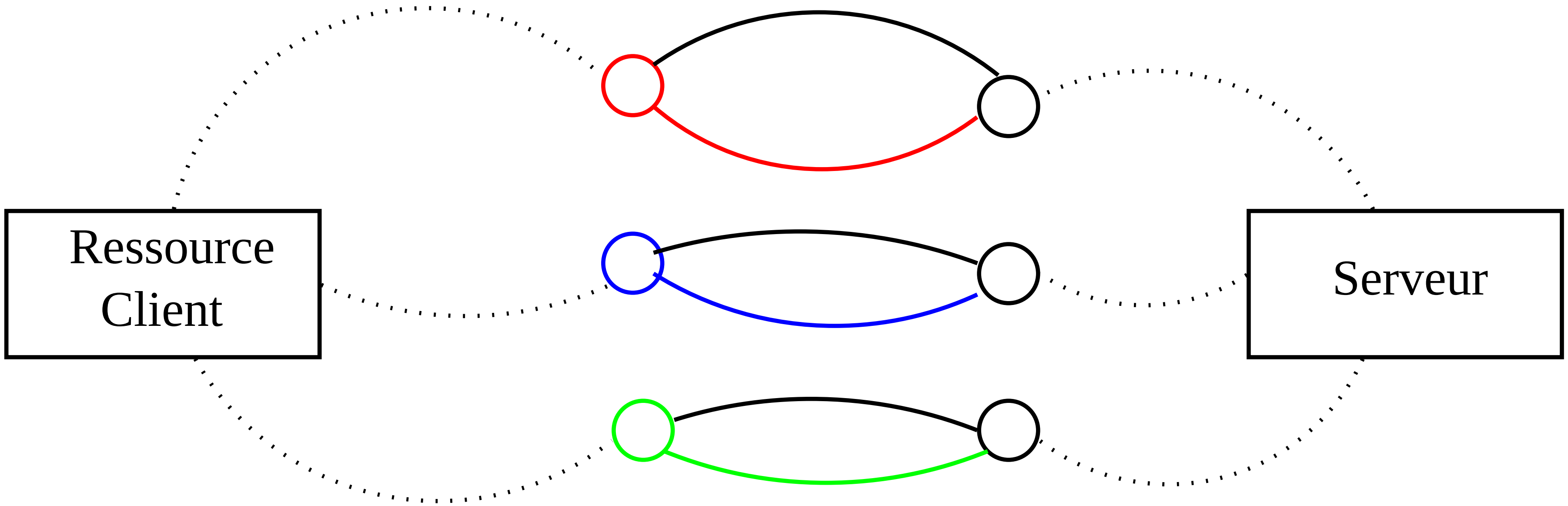
# Dynamic creation of agents



A connection:



# A network



## $\pi$-calculus: syntax

Let Name be an infinite set of channel names, and Label an infinite set of labels,

$$
\begin{array}{rcl}
P & ::= & \text{action}.P \\
& | & (P \mid P) \\
& | & (P + P) \\
& | & \emptyset
\end{array}
\qquad
\begin{array}{rcl}
\text{action} & ::= & c!^{i}[x_1, \ldots, x_n] \\
& | & c?^{i}[x_1, \ldots, x_n] \\
& | & *c?^{i}[x_1, \ldots, x_n] \\
& | & (\nu\ x) \\
& | & [x \diamond^{i} y]
\end{array}
$$

where $n \geqslant 0$, $c$, $x_1$, ..., $x_n$, $x$, $\in$ Name , $i \in$ Label, $\diamond \in \{=; \neq\}$,

$\nu$ and $?$ are the only name binders.
We denote by $fn(P)$ the set of free names in $P$, and by $bn(P)$ the set of bound names in $P$.

# Transition semantics

A reduction relation and a congruence relation give the semantics of the $\pi$-calculus:

- the reduction relation specifies the result of process computations:

$$
\begin{array}{rcll}
c?^{i}[\overline{y}]Q \mid c!^{j}[\overline{x}]P & \xrightarrow{i,j} & Q[\overline{y} \leftarrow \overline{x}] \mid P & \\
*c?^{i}[\overline{y}]Q \mid c!^{j}[\overline{x}]P & \xrightarrow{i,j} & Q[\overline{y} \leftarrow \overline{x}] \mid *c?^{i}[\overline{y}]Q \mid P & \\
P+Q & \xrightarrow{\varepsilon} & P & \\
P+Q & \xrightarrow{\varepsilon} & Q & \\
[x \diamond^{i} y].P & \xrightarrow{\varepsilon} & P & \text{when } x \diamond y, \diamond \in \{=, \neq\}
\end{array}
$$

$$
\frac{P \rightarrow Q}{(\nu\ x)P \rightarrow (\nu\ x)Q} \quad \frac{P' \equiv P \quad P \rightarrow Q \quad Q \equiv Q'}{P' \rightarrow Q'} \quad \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}
$$

# Congruence relation

- the congruence relation reveals redexs:

$$
\begin{array}{rcll}
P \mid Q & \equiv & Q \mid P & & \text{(Commutativity)} \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & & \text{(Associativity)} \\
(\nu\ x)P & \equiv & (\nu\ y)P[x \leftarrow y] & \text{if } y \notin fn(P) & (\alpha\text{-conversion}) \\
(\nu\ x)(\nu\ y)P & \equiv & (\nu\ y)(\nu\ x)P & & \text{(Swapping)} \\
((\nu\ x)P) \mid Q & \equiv & (\nu\ x)(P \mid Q) & \text{if } x \notin fn(Q) & \text{(Extrusion)} \\
(\nu\ x)P & \equiv & P & \text{if } x \notin fn(P) & \text{(Garbage collection)}
\end{array}
$$

# Exporting a channel

$$(\nu\ a)((\nu\ x)(a?[y].P(x,y)|(\nu\ y)(\nu\ x)a![x].R(x,y)))$$

$\equiv$ ( $\alpha$-conversion, swapping and extrusion)

$$(\nu\ a)(\nu\ x_1)(\nu\ x_2)(\nu\ y)(a?[y].P(x_1,y)|a![x_2].R(x_2,y))$$

$\longrightarrow$

$$(\nu\ a)(\nu\ x_1)(\nu\ x_2)(\nu\ y)(P(x_1,x_2)|R(x_2,y))$$

$\equiv$ (swapping and extrusion)

$$(\nu\ a)(\nu\ x_2)((\nu\ x_1)P(x_1,x_2)|(\nu\ y)R(x_2,y))$$

# Example: syntax

$$\mathcal{S} := (\nu \text{ port})(\nu \text{ gen})$$
$$(\text{Server} \mid \text{Customer} \mid \text{gen}!^{0}[])$$

where

$$\text{Server} := *\text{port}?^{1}[\text{info},\text{add}](\text{add}!^{2}[\text{info}])$$

$$\text{Customer} := *\text{gen}?^{3}[]\ ((\nu \text{ data})\ (\nu \text{ email})$$
$$(\text{port}!^{4}[\text{data}, \text{email}] \mid \text{gen}!^{5}[]))$$

# Example: computation

$$(\nu\ \mathrm{port})(\nu\ \mathrm{gen})$$
$$(\mathrm{Server} \mid \mathrm{Customer} \mid \mathrm{gen}!^{0}[])$$

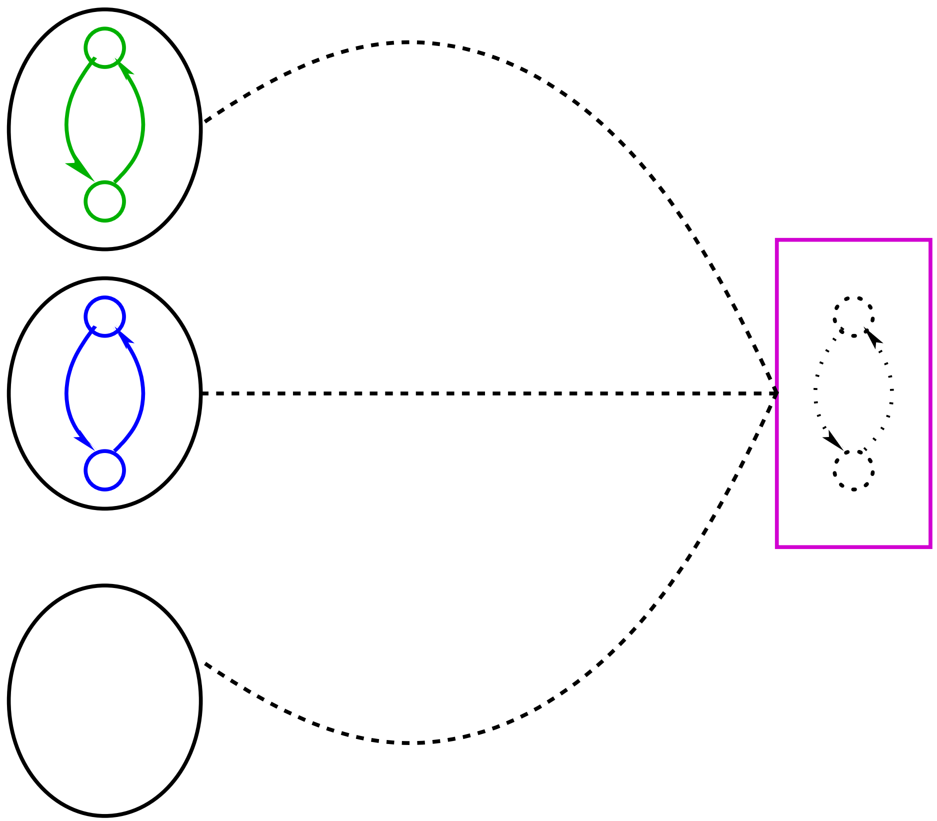
$$\xrightarrow{3,0} (\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ \mathrm{data}_1)(\nu\ \mathrm{email}_1)$$
$$(\mathrm{Server} \mid \mathrm{Customer} \mid \mathrm{gen}!^{5}[] \mid \mathrm{port}!^{4}[\mathrm{data}_1,\mathrm{email}_1])$$

$$\xrightarrow{1,4} (\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ \mathrm{data}_1)(\nu\ \mathrm{email}_1)$$
$$(\mathrm{Server} \mid \mathrm{Customer} \mid \mathrm{gen}!^{5}[] \mid \mathrm{email}_1!^{2}[\mathrm{data}_1])$$

$$\xrightarrow{3,5} (\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ \mathrm{data}_1)(\nu\ \mathrm{email}_1)(\nu\ \mathrm{data}_2)(\nu\ \mathrm{email}_2)$$
$$(\mathrm{Server} \mid \mathrm{Customer} \mid \mathrm{gen}!^{5}[] \mid \mathrm{email}_1!^{2}[\mathrm{data}_1] \mid \mathrm{port}!^{4}[\mathrm{data}_2,\mathrm{email}_2])$$

$$\xrightarrow{1,4} (\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ \mathrm{data}_1)(\nu\ \mathrm{email}_1)(\nu\ \mathrm{data}_2)(\nu\ \mathrm{email}_2)$$
$$(\mathrm{Server} \mid \mathrm{Customer} \mid \mathrm{gen}!^{5}[] \mid \mathrm{email}_1!^{2}[\mathrm{data}_1] \mid \mathrm{email}_2!^{2}[\mathrm{data}_2])$$

# Example: a server



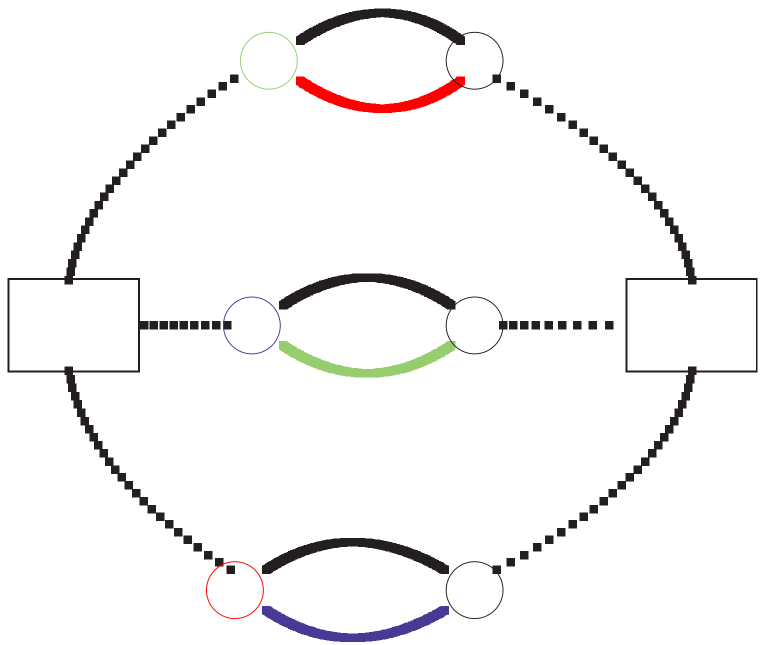
# $\alpha$-conversion

$\alpha$-conversion destroys the link between names and processes which have declared them:

$$
\begin{array}{l}
(\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ \mathrm{data}_1)(\nu\ \mathrm{email}_1) \\
(\nu\ \mathrm{data}_2)(\nu\ \mathrm{email}_2) \\
\ (\mathrm{Server} \mid \mathrm{Customer} \mid \mathrm{gen}!^{5}[] \\
\ \mid \mathrm{email}_1!^{4}[\mathrm{data}_1] \mid \mathrm{email}_2!^{4}[\mathrm{data}_2])
\end{array}
$$

$\sim_\alpha$

$$
\begin{array}{l}
(\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ \mathrm{data}_2) \\
(\nu\ \mathrm{email}_1)(\nu\ \mathrm{data}_1)(\nu\ \mathrm{email}_2) \\
\ (\mathrm{Server} \mid \mathrm{Customer} \mid \mathrm{gen}!^{5}[] \\
\ \mid \mathrm{email}_1!^{4}[\mathrm{data}_2] \mid \mathrm{email}_2!^{4}[\mathrm{data}_1])
\end{array}
$$



# Other forms of Mobility

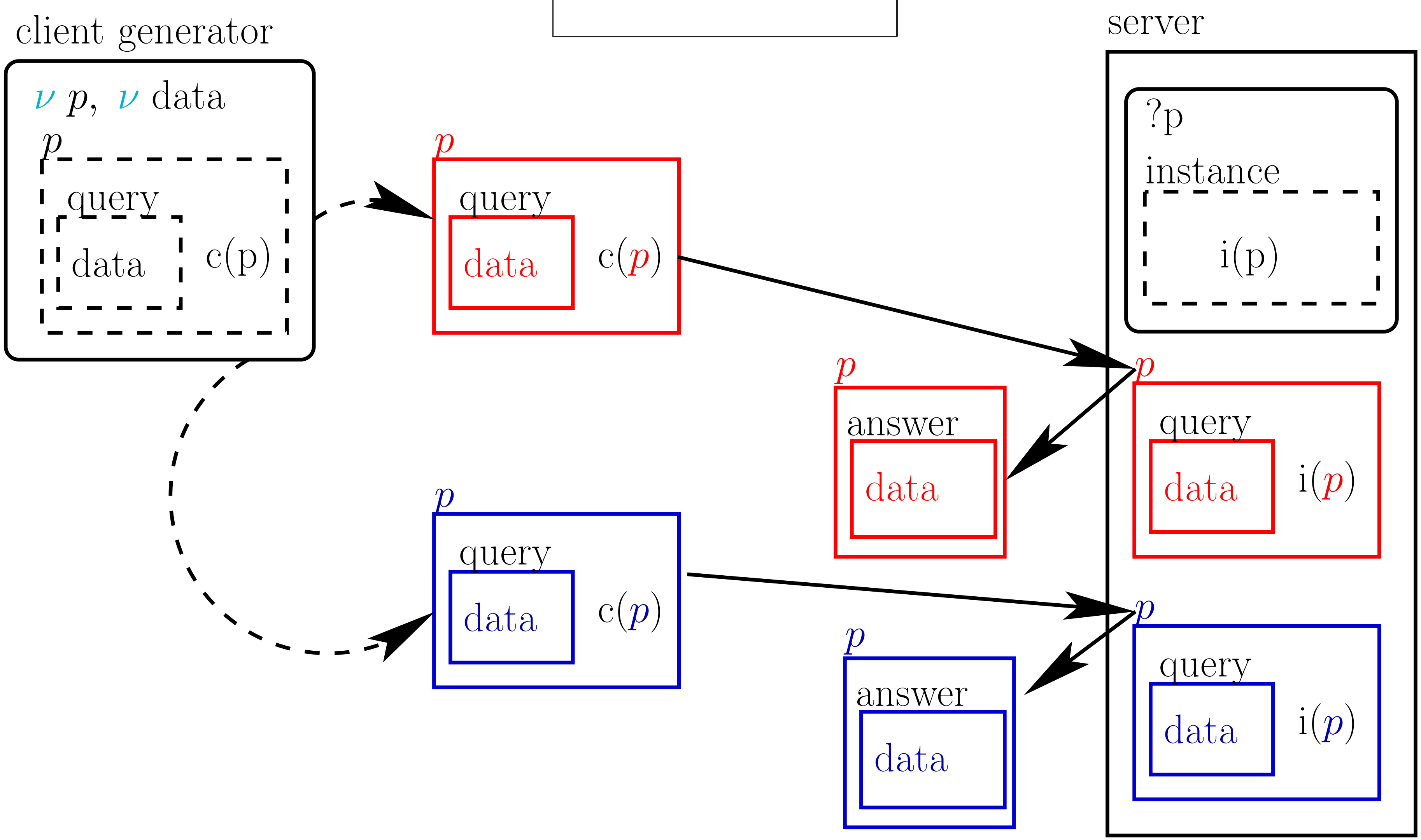
## Mobile Ambients

Ambients are named boxes containing other ambients (and/or) some agents.

Agents:

- provide capabilities to their surrounding ambients for local migration and other ambient dissolution;
- dynamically create new ambients, names and agents;
- communicate names to each others.

# An ftp-server



# Syntax

Let Name be an infinite countable set of ambient names and Label an infinite countable set of labels.

$$
\begin{array}{lll}
n & \in \mathrm{Name} & \text{(ambient name)} \\
l & \in \mathrm{Label} & \text{(label)}
\end{array}
$$

$$
\begin{array}{lcll}
P, Q & ::= & (\nu\ n)P & \text{(restriction)} \\
 & \mid & \mathbf{0} & \text{(inactivity)} \\
 & \mid & P \mid Q & \text{(composition)} \\
 & \mid & n^{l}[P] & \text{(ambient)} \\
 & \mid & M & \text{(capability action)} \\
 & \mid & \mathrm{io} & \text{(input/output action)}
\end{array}
$$

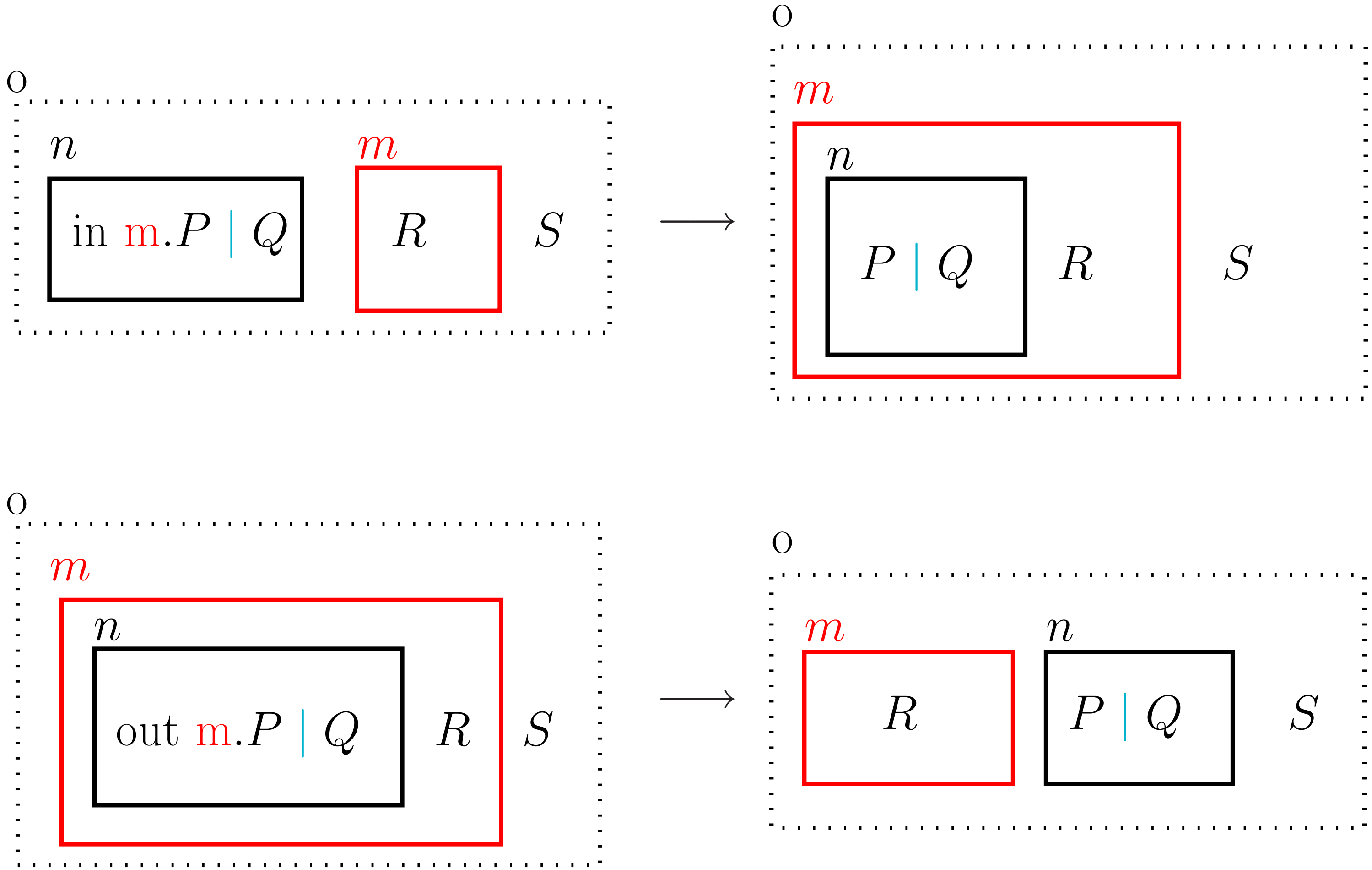
# Capability and actions

$$
\begin{array}{rcll}
M & ::= & \mathrm{in}^{l}\ n.P & \text{(can enter an ambient named } n\text{)} \\
 & | & \mathrm{out}^{l}\ n.P & \text{(can exit an ambient named } n\text{)} \\
 & | & \mathrm{open}^{l}\ n.P & \text{(can open an ambient named } n\text{)} \\
 & | & !\mathrm{open}^{l}\ n.P & \text{(can open several ambients named } n\text{)}
\end{array}
$$

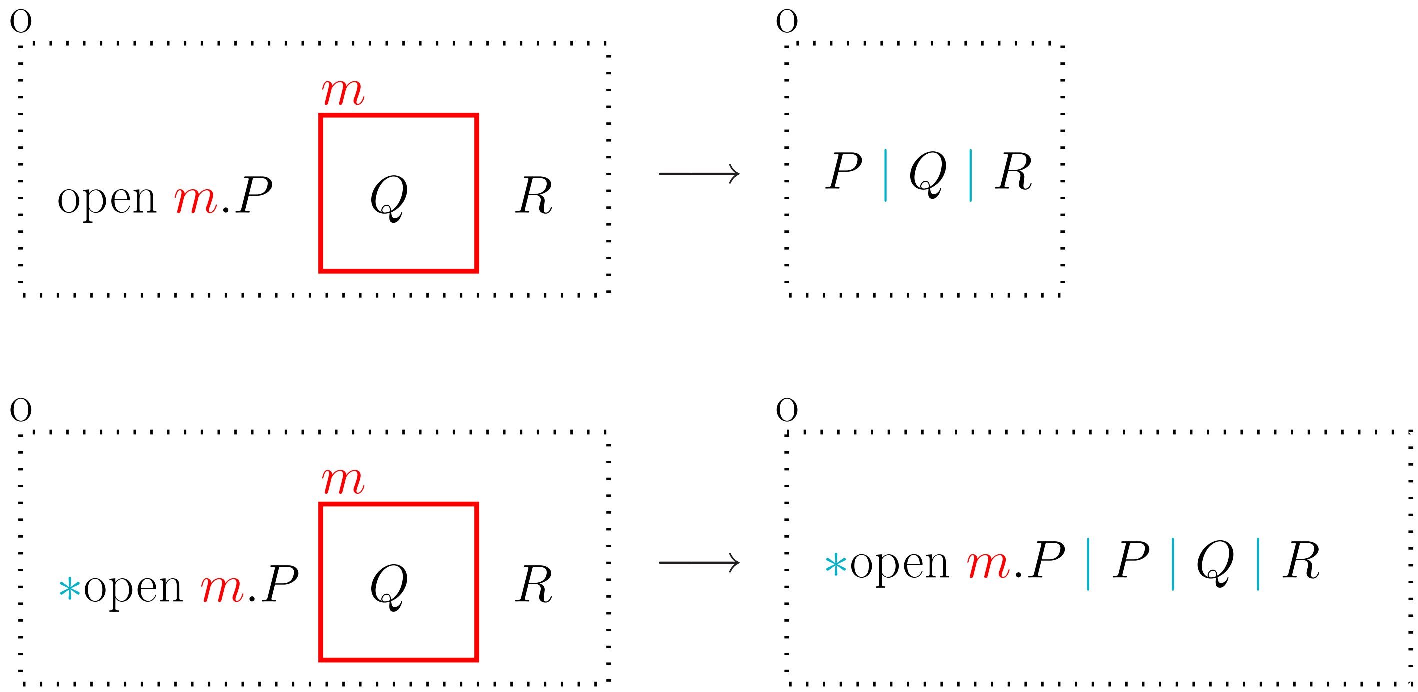
$$
\begin{array}{rcll}
\mathrm{io} & ::= & (n)^{l}.P & \text{(input action)} \\
 & | & !(n)^{l}.P & \text{(input action with replication)} \\
 & | & \langle n \rangle^{l} & \text{(async output action)}
\end{array}
$$

The only name binders are $(\nu\ \_)$, $(\_)$ and $!(\_)$.

# Ambient Migration



# Ambient Dissolution



# Communication

$$\langle m\rangle \mid (n).P \mid Q \longrightarrow P_{[n\leftarrow m]} \mid Q$$

$$\langle m\rangle \mid *(n).P \mid Q \longrightarrow *(n).P \mid P_{[n\leftarrow m]} \mid Q$$

# An ftp-server

$$\mathcal{S} := (\nu \mathrm{Pub})(\mathrm{S} \mid !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{21})$$

*where*

$$\mathrm{Pub} := (\nu\ \mathrm{request})(\nu\ \mathrm{make})(\nu\ \mathrm{server})(\nu\ \mathrm{duplicate})(\nu\ \mathrm{instance})(\nu\ \mathrm{answer}),$$

$$\mathrm{C} := (\nu\ q)(\nu\ p)p^{12}[\mathrm{C}_1 \mid \mathrm{C}_2 \mid \mathrm{C}_3] \mid \langle \mathrm{make} \rangle^{20},$$
$$\mathrm{C}_1 := \mathrm{request}^{13}[\langle q \rangle^{14}],\ \mathrm{C}_2 := \mathrm{open}^{15}\mathrm{instance},$$
$$\mathrm{C}_3 := \mathrm{in}^{16}\mathrm{server}.\mathrm{duplicate}^{17}[\mathrm{out}^{18}p.\langle p \rangle^{19}],$$

$$\mathrm{S} := \mathrm{server}^{1}[\mathrm{S}_1 \mid \mathrm{S}_2],\ \mathrm{S}_1 := !\mathrm{open}^{2}\mathrm{duplicate},\ \mathrm{S}_2 := !(k)^{3}.\mathrm{instance}^{4}[\mathrm{I}],$$

$$\mathrm{I} := \mathrm{in}^{5}k.\mathrm{open}^{6}\mathrm{request}.(\mathrm{rep})^{7}(\mathrm{I}_1 \mid \mathrm{I}_2),\ \mathrm{I}_1 := \mathrm{answer}^{8}[\langle \mathrm{rep} \rangle^{9}],\ \mathrm{I}_2 := \mathrm{out}^{10}\mathrm{server}.$$

$$(\nu \mathrm{Pub})(\mathrm{S} \mid !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{21})$$

$$\rightarrow$$

$$(\nu \mathrm{Pub}) \left( \begin{array}{l} !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1}[\mathrm{S}_1 \mid \mathrm{S}_2] \mid \\ (\nu\ q_1)(\nu\ p_1) p_1{}^{12} \left[ \begin{array}{l} \mathrm{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathrm{C}_2 \mid \\ \mathrm{in}^{16}\mathrm{server}.\mathrm{duplicate}^{17}[\mathrm{out}^{18} p_1.\langle p_1 \rangle^{19}] \end{array} \right] \end{array} \right)$$

$$\rightarrow$$

$$(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1)$$

$$\left( !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1} \left[ \mathrm{S}_1 \mid \mathrm{S}_2 \mid p_1{}^{12} \left[ \begin{array}{l} \mathrm{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathrm{C}_2 \mid \\ \mathrm{duplicate}^{17}[\mathrm{out}^{18} p_1.\langle p_1 \rangle^{19}] \end{array} \right] \right] \right)$$

$$\rightarrow$$

$$(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1)$$

$$\left( !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1} \left[ \begin{array}{l} !\mathrm{open}^{2}\mathrm{duplicate} \mid \mathrm{S}_2 \mid \mathrm{duplicate}^{17}[\langle p_1 \rangle^{19}] \mid \\ p_1{}^{12}[\mathrm{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathrm{C}_2] \end{array} \right] \right)$$

$$(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1)$$
$$\left( !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1} \left[ \begin{array}{l} !\mathrm{open}^{2}\mathrm{duplicate} \mid \mathrm{S}_2 \mid \mathrm{duplicate}^{17}[\langle p_1 \rangle^{19}] \mid \\ p_1{}^{12}[\mathrm{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathrm{C}_2] \end{array} \right] \right)$$
$$\rightarrow$$
$$(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1)$$
$$\left( !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1} \left[ \begin{array}{l} \mathrm{S}_1 \mid !(k)^{3}.\mathrm{instance}^{4}[\mathrm{I}] \mid \langle p_1 \rangle^{19} \mid \\ p_1{}^{12}[\mathrm{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathrm{C}_2] \end{array} \right] \right)$$
$$\rightarrow$$
$$(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1)$$
$$\left( !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1} \left[ \begin{array}{l} \mathrm{S}_1 \mid \mathrm{S}_2 \mid p_1{}^{12}[\mathrm{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathrm{C}_2] \mid \\ \mathrm{instance}^{4}[\mathrm{in}^{5} p_1.\mathrm{open}^{6}\mathrm{request}.(\mathrm{rep})^{7}(\mathrm{I}_1 | \mathrm{I}_2)] \end{array} \right] \right)$$
$$\rightarrow$$
$$(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1)$$
$$\left( !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1} \left[ \begin{array}{l} \mathrm{S}_1 \mid \mathrm{S}_2 \mid \\ p_1{}^{12} \left[ \begin{array}{l} \mathrm{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathrm{open}^{15}\mathrm{instance} \mid \\ \mathrm{instance}^{4}[\mathrm{open}^{6}\mathrm{request}.(\mathrm{rep})^{7}(\mathrm{I}_1 | \mathrm{I}_2)] \end{array} \right] \end{array} \right] \right)$$

$$
\begin{array}{l}
(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1) \\
\left( !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1} \left[ \begin{array}{l} \mathrm{S}_1 \mid \mathrm{S}_2 \mid \\ p_1{}^{12} \left[ \begin{array}{l} \mathrm{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathrm{open}^{15}\mathrm{instance} \mid \\ \mathrm{instance}^{4}[\mathrm{open}^{6}\mathrm{request}.(\mathrm{rep})^{7}(\mathrm{I}_1 \mid \mathrm{I}_2)] \end{array} \right] \end{array} \right] \right)
\end{array}
$$

$$\longrightarrow$$

$$
\begin{array}{l}
(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1) \\
\left( !(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1} \left[ \mathrm{S}_1 \mid \mathrm{S}_2 \mid p_1{}^{12} \left[ \begin{array}{l} \mathrm{request}^{13}[\langle q_1 \rangle^{14}] \mid \\ \mathrm{open}^{6}\mathrm{request}.(\mathrm{rep})^{7}(\mathrm{I}_1 \mid \mathrm{I}_2) \end{array} \right] \right] \right)
\end{array}
$$

$$\longrightarrow^{*}$$

$$
\begin{array}{l}
(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1) \\
(!(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1}[\mathrm{S}_1 \mid \mathrm{S}_2 \mid p_1{}^{12}[\mathrm{answer}^{8}[\langle q_1 \rangle^{9}] \mid \mathrm{out}^{10}\mathrm{server}]])
\end{array}
$$

$$\longrightarrow$$

$$
\begin{array}{l}
(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1) \\
(!(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1}[\mathrm{S}_1 \mid \mathrm{S}_2] \mid p_1{}^{12}[\mathrm{answer}^{8}[\langle q_1 \rangle^{9}]])
\end{array}
$$

$$\longrightarrow^{*}$$

$$
\begin{array}{r}
(\nu \mathrm{Pub})(\nu\ q_1)(\nu\ p_1)(\nu\ q_2)(\nu\ p_2)(!(x)^{11}.C \mid \langle \mathrm{make} \rangle^{20} \mid \mathrm{server}^{1}[\mathrm{S}_1 \mid \mathrm{S}_2] \\
\mid p_1{}^{12}[\mathrm{answer}^{8}[\langle q_1 \rangle^{9}]] \mid p_2{}^{12}[\mathrm{answer}^{8}[\langle q_2 \rangle^{9}]])
\end{array}
$$

## “Restricted” mobility

In the $\pi$-calculus, when a message is sent, we cannot statically know where this message will be received, because when a channel name is communicated to a process, this process potentially get the capability to send and to receive on this channel.
In many languages and formalisms, only the capability to send over a channel is communicated. Only the process that has declared the channel name can received infornation through this channel.
This helps in asynchronous implementation, and syntactily restricts the potential attack over the system.

# Join-calculus

Let Name be an infinite set of channel names,

$$
\begin{array}{rcll}
\mathrm{P} & ::= & c![x_1, ..., x_n] & \text{(output)} \\
& | & \mathrm{P} \mid \mathrm{P} & \text{(parallel composition)} \\
& | & \mathrm{def}\ \mathrm{J} \triangleright \mathrm{P}\ \mathrm{in}\ \mathrm{P} & \text{(definition)}
\end{array}
$$

$$
\begin{array}{rcll}
\mathrm{J} & ::= & c?[x_1, ..., x_n] & \text{(message pattern)} \\
& | & \mathrm{J} \mid \mathrm{J} & \text{(join patterns)}
\end{array}
$$

where $n \geqslant 0$, $c$, $x_1$, ..., $x_n$, $x$, $\in$ Name , definitions are the only name binders. For instance, the definition:

$$P = \mathrm{def}\ c?[x_1, ..., x_n] \mid \mathrm{J} \triangleright P_1\ \mathrm{in}\ P_2$$

binds the name $c$ in the process $P$ and the names $x_i$ in the process $P_1$.

## Operational semantics

A state is given by a set of definitions over distinct names and a process.
The transition system is defined as follows:

1. the system may discover new definitions:

$$(D, (\mathrm{def}\ \mathrm{J} \triangleright P_1 \text{ in } P_2) | P_3) \rightarrow (D \uplus \{\mathrm{def}\ J \triangleright P_1\}, P_2 \mid P_3)$$

2. the system may perform communications:

$$(D, c_1![\overline{x}_1] \mid \ \dots \ \mid c_n![\overline{x}_n] \mid Q) \rightarrow (D, Q \mid P\{\overline{y}_i \leftarrow \overline{x}_i\})$$
$$\text{when } (\mathrm{def}\ c_1?[\overline{y}_1] | \ \dots \ | c_n?[\overline{y}_n] \triangleright P) \in D$$

## Encoding the $\pi$-calculus: Intuition

- In $\pi$-calculus, a communication can be performed each time both an emmission and a reception is made over the same channel name:

  so to each $\pi$-channel name $c$, we will associate a join-definition that waits for simultaneous emission over two names, the first one $c_i$ describes receptions over $c$ and the second one $c_o$ describes emissions.

- We then use a continuation-style, to encode $\pi$-processes:

  to each $\pi$-process we associate a join-definition (only used once), that introduces a fresh name (which must be seen as a pointer to the continuation), and send this continuation with information about communicated names to the definition dealing with the channel the action is performed on. In presence of the good co-action, the channel definition returns a message to the continuation-definition with information about communicated channel names.

## Encoding the $\pi$-calculus

$$
\begin{array}{rcl}
[\![(\nu\ x)P]\!] & = & \mathrm{def}\ c_i?[\mathrm{cont}_i] \mid c_o?[\mathrm{cont}_o, x_i, x_o] \triangleright (\mathrm{cont}_i![x_i, x_o] \mid \mathrm{cont}_o![])\ \mathrm{in}\ [\![P]\!] \\
[\![c?[x].P]\!] & = & \mathrm{def}\ \mathrm{cont}?[x_i, x_o] \triangleright [\![P]\!]\ \mathrm{in}\ c_i![\mathrm{cont}] \\
[\![c![x].P]\!] & = & \mathrm{def}\ \mathrm{cont}?[] \triangleright [\![P]\!]\ \mathrm{in}\ c_o![\mathrm{cont}, x_i, x_o]
\end{array}
$$

# Example

$$(\nu\ x)(x![a] \mid x![b] \mid x?[u].y![u])$$

$$\left(\emptyset, \begin{array}{l} \text{def } x_o?[v_i, v_o] \mid x_i?[\text{cont}] \triangleright \text{cont}![v_i, v_o] \text{ in} \\ x_o![a_i, a_o] \mid x_o![b_i, b_o] \mid \text{def cont}?[u_i, u_o] \triangleright y_o![u_i, u_o] \text{ in } x_i![\text{cont}] \end{array}\right)$$

$$\rightarrow^* \left(\left\{ \begin{array}{l} \text{def } x_o?[v_i, v_o] \mid x_i?[\text{cont}] \triangleright \text{cont}![v_i, v_o], \\ \text{def cont}?[u_i, u_o] \triangleright y_o![u_i, u_o] \end{array} \right\}, x_o![a_i, a_o] \mid x_o![b_i, b_o] \mid x_i![\text{cont}]\right)$$

$$\rightarrow \left(\left\{ \begin{array}{l} \text{def } x_o?[v_i, v_o] \mid x_i?[\text{cont}] \triangleright \text{cont}![v_i, v_o], \\ \text{def cont}?[u_i, u_o] \triangleright y_o![u_i, u_o] \end{array} \right\}, x_o![b_i, b_o] \mid \text{cont}![a_i, a_o]\right)$$

$$\rightarrow \left(\left\{ \begin{array}{l} \text{def } x_o?[v_i, v_o] \mid x_i?[\text{cont}] \triangleright \text{cont}![v_i, v_o], \\ \text{def cont}?[u_i, u_o] \triangleright y_o![u_i, u_o] \end{array} \right\}, x_o![b_i, b_o] \mid y_o![a_i, a_o]\right)$$

# Reference

This join-calculus part has deeply been inspired from Cedric Fournet's thesis.

# A shared-memory example

# Motivation

We want to describe in the $\pi$-calculus a shared-memory in which:

- each process can allocate new cells,
- each authorized process can read the content of a cell,
- each authorized process can write inside a cell, overwriting the former content.

## Specification

A memory cell will be denoted by three channel names, cell, read, write: :

- a channel name cell describes the content of the cell:

  the process cell![data] means that the cell cell contains the information data, this name is internal to the memory (not visible by the user).

- a channel name read allows reading requests:

  the process read![port] is a request to read the content of the cell, and send it to the port port,

- a channel name write allows writing requests:

  the process write![data] is a request to write the information data inside the cell.

# Shared Memory

$\text{System} := (\nu\ \text{create})(\nu\ \text{null})(*\text{create}?[\text{d}].\text{Allocate}(\text{d}))$

$$\begin{aligned}&\text{Allocate}(\text{d}) :=\\ &\qquad (\nu\ \text{cell})(\nu\ \text{write})(\nu\ \text{read})\\ &\qquad\quad (\text{init}(\text{cell}) \mid \text{read}(\text{read},\text{cell}) \mid \text{write}(\text{write},\text{cell}) \mid \text{d}![\text{read};\text{write}])\end{aligned}$$

where

- $\text{init}(\text{cell}) := \text{cell}![\text{null}]$
- $\text{read}(\text{read},\text{cell}) := *\text{read}?[\text{port}].\text{cell}?[\text{u}].(\text{cell}![\text{u}] \mid \text{port}![\text{u}])$
- $\text{write}(\text{write},\text{cell}) := *\text{write}?[\text{data}].\text{cell}?[\text{u}].\text{cell}![\text{data}]$

# Trace example

$(\nu\ \text{create})(\nu\ \text{null})$
$\quad(*\text{create}?[\text{d}].\text{Allocate}(\text{d})$
$\quad\mid (\nu\ \text{address})(\nu\ \text{data})\text{create}![\text{address}].\text{address}?[\text{r};\text{w}].\text{w}![\text{data}].\text{r}![\text{address}])$

$\longrightarrow$

$(\nu\ \text{create})\ (\nu\ \text{null})\ (\nu\ \text{cell})\ (\nu\ \text{write})\ (\nu\ \text{read})\ (\nu\ \text{address})\ (\nu\ \text{data})$
$\quad(\ *\text{create}?[\text{d}].\text{Allocate}(\text{d})\ \mid\ \text{read}(\text{read},\text{cell})\ \mid\ \text{write}(\text{write},\text{cell})$
$\quad\mid\ \text{cell}![\text{null}]\ \mid\ \text{address}![\text{read},\text{write}]\ \mid\ \text{address}?[\text{r};\text{w}].\text{w}![\text{data}].\text{r}![\text{address}])$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\text{create}?[\text{d}].\text{Allocate}(\text{d})\ \mid\ \text{read}(\text{read},\text{cell})\ \mid\ \text{write}(\text{write},\text{cell})$
$\qquad\mid\ \text{cell}![\text{null}]\ \mid\ \text{write}![\text{data}].\text{read}![\text{address}])$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\text{create}?[\text{d}].\text{Allocate}(\text{d})\ \mid\ \text{read}(\text{read},\text{cell})\ \mid\ \text{write}(\text{write},\text{cell})$
$\qquad\mid\ \text{cell}![\text{null}]\ \mid\ \text{cell}?[\text{u}].\text{cell}![\text{data}]\ \mid\ \text{read}![\text{address}])$

# Expected Derivation

$(\nu\ \overline{c})(\ *\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})$
$\quad\mid\ \mathrm{cell}![\mathrm{null}]\ \mid\ \mathrm{cell}?[\mathrm{u}].\mathrm{cell}![\mathrm{data}]\ \mid \mathrm{read}![\mathrm{address}])$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})$
$\quad\mid\ \mathrm{cell}![\mathrm{data}] \mid \mathrm{read}![\mathrm{address}])$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})$
$\quad\mid\ \mathrm{cell}![\mathrm{data}] \mid \mathrm{address}![\mathrm{data}])$

# Unexpected Derivation

$$(\nu\ \overline{c})(\ *\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})$$
$$\mid\ \mathrm{cell}![\mathrm{null}]\ \mid\ \mathrm{cell}?[\mathrm{u}].\mathrm{cell}![\mathrm{data}]\ \mid\ \mathrm{read}![\mathrm{address}])$$

$$\longrightarrow$$

$$(\nu\ \overline{c})(\ *\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})$$
$$\mid\ \mathrm{cell}![\mathrm{null}]\ \mid\ \mathrm{cell}?[\mathrm{u}].\mathrm{cell}![\mathrm{data}]\ \mid\ \mathrm{address}![\mathrm{null}])$$

$$\longrightarrow$$

$$(\nu\ \overline{c})(\ *\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})$$
$$\mid\ \mathrm{cell}![\mathrm{data}]\ \mid\ \mathrm{address}![\mathrm{null}])$$

## Enforcing synchronisation

$$\text{System} := (\nu\ \text{create})(\nu\ \text{null})(*\text{create}?[\text{d}].\text{Allocate}(\text{d}))$$

$$\begin{array}{l}\text{Allocate}(\text{d}) := \\ \qquad (\nu\ \text{cell})(\nu\ \text{write})(\nu\ \text{read}) \\ \qquad\quad \text{init}(\text{cell}) \ \mid\ \text{read}(\text{read,cell}) \ \mid\ \text{write}(\text{write,cell}) \ \mid\ \text{d}![\text{read;write}]\end{array}$$

where

- $\text{init}(\text{cell}) := \text{cell}![\text{null}]$
- $\text{read}(\text{read,cell}) := *\text{read}?[\text{port}].\text{cell}?[\text{u}](\text{cell}![\text{u}] \ \mid\ \text{port}![\text{u}])$
- $\text{write}(\text{write,cell}) := *\text{write}?[\text{data,ack}].\text{cell}?[\text{u}].(\text{cell}![\text{data}] \mid \text{ack}![])$

$(\nu$ create$)(\nu$ null$)$
(∗create?[d].Allocate(d)
| $(\nu$ address$)(\nu$ data$)(\nu$ ack$)$
create![address].address?[r;w].w![data;ack].ack?[].r![address])

$\longrightarrow$

$(\nu\ \overline{c})$( ∗create?[d].Allocate(d) | read(read,cell) | write(write,cell) | cell![null]
| address![read,write] | address?[r;w].w![data;ack].ack?[].r![address])

$\longrightarrow$

$(\nu\ \overline{c})$( ∗create?[d].Allocate(d) | read(read,cell) | write(write,cell)
| cell![null] | write![data;ack].ack?[].read![address])

$\longrightarrow$

$(\nu\ \overline{c})$( ∗create?[d].Allocate(d) | read(read,cell) | write(write,cell)
| cell![null] | cell?[u].(cell![data] | ack![])
| ack?[].read![address])

$$
\begin{aligned}
&(\nu\ \overline{c})(\ {*}\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})\\
&\qquad \mid\ \mathrm{cell}![\mathrm{null}]\ \mid\ \mathrm{cell}?[\mathrm{u}].(\mathrm{cell}![\mathrm{data}] \mid \mathrm{ack}![])\\
&\qquad \mid \mathrm{ack}?[].\mathrm{read}![\mathrm{address}])\\
&\longrightarrow\\
&(\nu\ \overline{c})(\ {*}\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})\\
&\qquad \mid\ (\mathrm{cell}![\mathrm{data}] \mid \mathrm{ack}![]) \mid \mathrm{ack}?[].\mathrm{read}![\mathrm{address}])\\
&\longrightarrow\\
&(\nu\ \overline{c})(\ {*}\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})\\
&\qquad \mid\ \mathrm{cell}![\mathrm{data}] \mid \mathrm{read}![\mathrm{address}])\\
&\longrightarrow\\
&(\nu\ \overline{c})(\ {*}\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(\mathrm{d})\ \mid\ \mathrm{read}(\mathrm{read},\mathrm{cell})\ \mid\ \mathrm{write}(\mathrm{write},\mathrm{cell})\\
&\qquad \mid\ \mathrm{cell}![\mathrm{data}] \mid\ \mathrm{address}![\mathrm{data}])
\end{aligned}
$$

# Using Mutex

$\text{System} := (\nu\ \text{create})(\nu\ \text{null})(*\text{create}?[\text{d}]\text{Allocate}(\text{d}))$

$\text{Allocate}(\text{d}) := (\nu\ \text{cell})(\nu\ \text{mutex})(\nu\ \text{nomutex})(\nu\ \text{write})(\nu\ \text{read})(\nu\ \text{lock})(\nu\ \text{unlock})$
$\quad \text{init}(\text{cell},\text{mutex})\ \mid\ \text{read}(\text{read},\text{cell})\ \mid\ \text{write}(\text{write},\text{cell})$
$\quad \mid\ \text{lock}(\text{lock},\text{mutex},\text{nomutex})\ \mid\ \text{unlock}(\text{unlock},\text{nomutex},\text{nomutex})$
$\quad \mid\ \text{d}![\text{read};\text{write};\text{lock};\text{unlock}]$

where

$\text{init}(\text{cell},\text{mutex}) := \text{cell}![\text{null}]\ \mid\ \text{mutex}![]$

$\text{read}(\text{read},\text{cell}) := *\text{read}?[\text{port}].\text{cell}?[\text{u}](\text{cell}![\text{u}]\ \mid\ \text{port}![\text{u}])$

$\text{write}(\text{write},\text{cell}) := *\text{write}?[\text{data},\text{ack}].\text{cell}?[\text{u}].(\text{cell}![\text{data}]\ \mid\ \text{ack}![])$

$\text{lock}(\text{lock},\text{mutex},\text{nomutex}) := *\text{lock}?[\text{ack}].\text{mutex}?[].(\text{ack}![]\ \mid\ \text{nomutex}![])$

$\text{unlock}(\text{unlock},\text{mutex},\text{nomutex}) := *\text{unlock}?[\text{ack}].\text{nomutex}?[].(\text{ack}![]\ \mid\ \text{mutex}![])$