

THE BoPi PROJECT

a *simple, distributed* implementation of
pi-calculus with *xml datatypes*

`www.cs.unibo.it/BoPi`

*Samuele Carpineti, Cosimo Laneve,
Paolo Milazzo, Lucian Wischik*

a pi calculus agent

```
(new tun@bo)(  
   $\overline{glob\ tun}$   
  |  $tun(x).x(y).\overline{y\ tun}$   
  
)  
|  $\overline{glob(z).\overline{z\ blq}}$   
|  $\overline{blq\ alert}$   
  
// create a fresh channel, at bo  
// extrude channel 'tun' over 'glob'  
// receive channel 'x', then receive 'y'  
// on 'x' and forward 'tun' on 'x'  
  
// receive channel 'z' and send 'blq' over it  
// send 'alert' over blq
```

about distribution:

- there are *three agents* which could be located in different hosts
- (*global consensus issue*) we may assume that channels are managed by a unique host (managing means that the host is responsible of the inputs of the channels it manages)
- *input capability* breaks the property of unique host for inputs . . .

input capability and distribution

the initial configuration is

bo:

```
(new tun@bo)(  
   $\overline{glob\ tun}$   
  /  $tun(x).x(y).\overline{y\ tun}$   
)
```

glob:

```
 $glob(z).\overline{z\ blq}$ 
```

blq:

```
 $\overline{blq\ alert}$ 
```

channels are located at IP addresses; inputs are loaded into the subjects' locations

... after some reduction ...

bo, tun:

```
 $blq(y).\overline{y\ tun}$ 
```

glob:

blq:

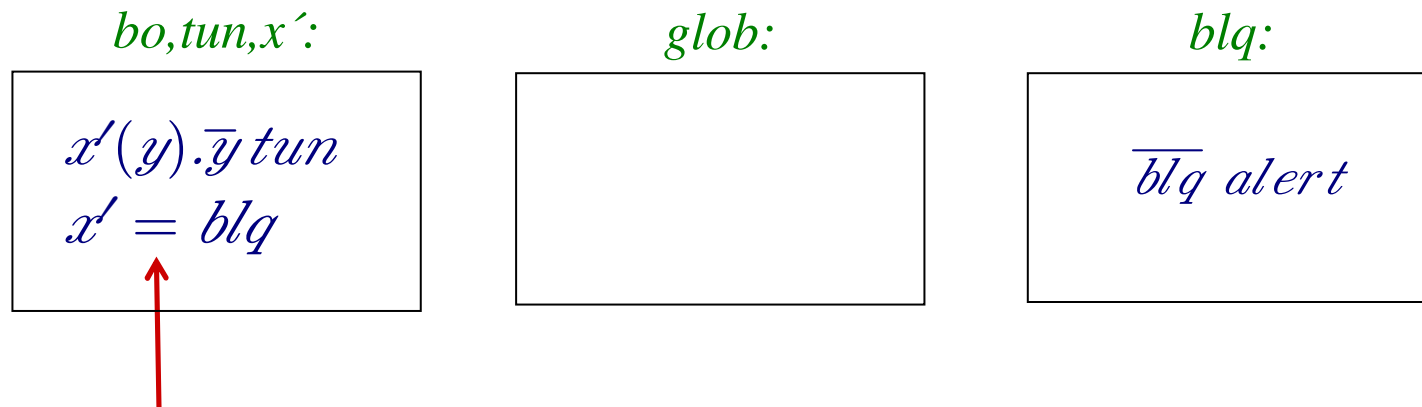
```
 $\overline{blq\ alert}$ 
```

*a new input (aka service) has been created at a **wrong location***

input capability, distribution, and fusions

- *input capability is hard to implement: never use it!*
- *cf. join-calculus, local pi*
- *first solution (CONCUR'02) use fusions instead*
 - *bound inputs are renamed with fresh local names*
 - *the fresh name is fused with the received name*

... after some reduction ...



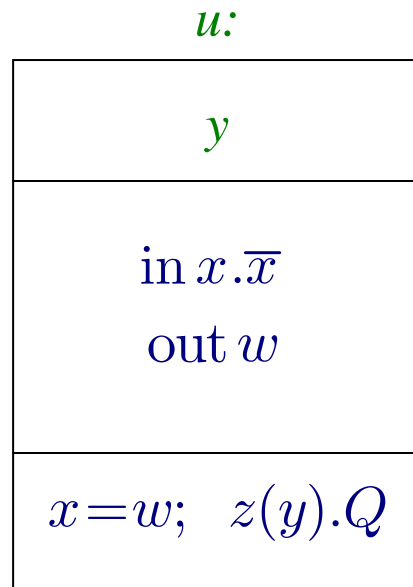
an explicit fusion between two names

the run-time support of the language should solve

the explicit fusion calculus

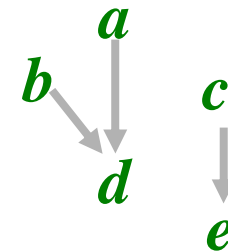
“An *explicit fusion* is something that allows two names to be used *interchangeably*”

the fusion machine



← ***fusion pointer***, so
any atom can migrate
from here to y .

Collectively, the
fusion pointers make
a *forest* which
respects a total order
on names:



the fusion machine: **problems**

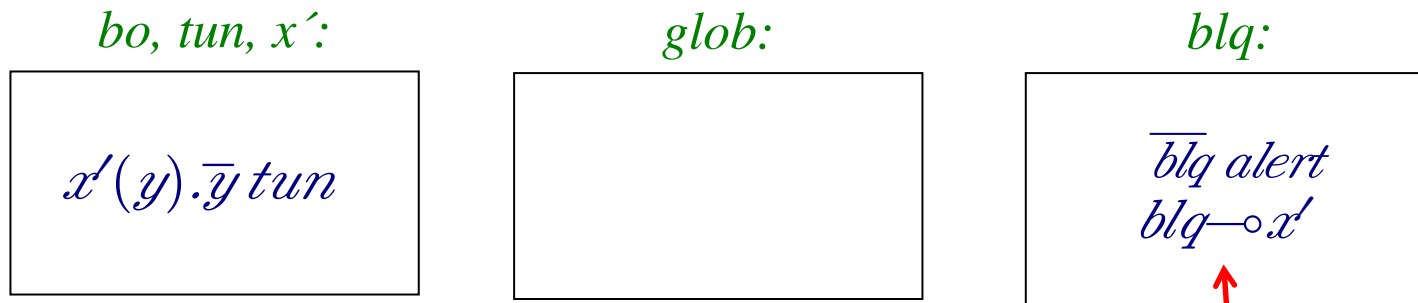
- *fusions are implemented with persistent tree of forwarders*
- *these trees are awkward to manage (cf. optimizations)*
- *these trees are fragile in presence of failures*
- *the management of continuations was difficult (compilation into solos)*

robustness of the machine has been the main reason for pushing the redesign on

input capability, distribution, and **linear forwarders**

- *input capability is hard to implement: never use it!*
- *second solution (CONCUR'03) use **linear forwarders** instead*

... after some reduction ...



*a **linear forwarder**
to be used exactly once*

the linear forwarder calculus

processes P

$$P ::= \mathbf{0} \quad | \quad \bar{x}\tilde{y} \quad | \quad x(\tilde{y}).P \quad | \quad (x)P \quad | \quad P/P \quad | \quad !P \quad | \quad x \multimap y$$

with *no input capability constraint*: in $x(u).P$, P has no input with u in subject position

structural congruence

the smallest equivalence closed wrt contexts, alpha-renaming, monoidal laws of $/$ with identity $\mathbf{0}$, and the laws:

$$!P \equiv P!P \quad (x)(y)P \equiv (y)(x)P \quad (x)(P/Q) \equiv P/(x)Q \text{ if } x \notin \text{fn } P$$

reduction the smallest relation satisfying the following and closed under \equiv , $(x)_-$ and $-/_-$.

$$u(\tilde{x}).P/\bar{u}\tilde{y} \quad \rightarrow \quad P\{\tilde{y}/\tilde{x}\}$$

$$\bar{x}\tilde{u}/x \multimap y \quad \rightarrow \quad \bar{y}\tilde{u}$$

the encoding of pi

$$\llbracket x(\tilde{y}).P \rrbracket_{\tilde{u}} = \begin{cases} x(\tilde{y}).\llbracket P \rrbracket_{\tilde{u}\tilde{y}} & \text{if } x \notin \tilde{u} \\ (\mathcal{U}'_i)(u_i \multimap \mathcal{U}'_i / \mathcal{U}'_i(\tilde{y}).\llbracket P \rrbracket_{\tilde{u}\tilde{y}}) & \text{if } x = u_i \end{cases}$$

$$\llbracket (x)P \rrbracket_{\tilde{u}} = (x)(\llbracket P \rrbracket_{\tilde{u}})$$

$$\llbracket P/Q \rrbracket_{\tilde{u}} = \llbracket P \rrbracket_{\tilde{u}} / \llbracket Q \rrbracket_{\tilde{u}}$$

$$\llbracket !P \rrbracket_{\tilde{u}} = !\llbracket P \rrbracket_{\tilde{u}}$$

$$\llbracket \bar{x}\tilde{y} \rrbracket_{\tilde{u}} = \bar{x}\tilde{y}$$

$$\llbracket \mathbf{0} \rrbracket_{\tilde{u}} = \mathbf{0}$$

an example

$$\begin{aligned} \llbracket \bar{u}y / u(x).P \rrbracket_u &= \bar{u}y / (\mathcal{U}')(\mathcal{U}' \multimap \mathcal{U}' / \mathcal{U}'(x).\llbracket P \rrbracket_{xu}) \\ &\rightarrow (\mathcal{U}')(\bar{u}'y / \mathcal{U}'(x).\llbracket P \rrbracket_{xu}) \\ &\rightarrow (\mathcal{U}')(\llbracket P \rrbracket_{xu}\{y/x\}) \\ &\equiv \llbracket P \rrbracket_{xu}\{y/x\} \end{aligned}$$

remarks on the encoding of pi

1. pi contexts are exactly as discriminating as linear-forwarder contexts:

Theorem. For pi calculus terms P and Q , then $P \approx Q$ iff $[[P]] \approx [[Q]]$

2. linearity plays a crucial role for preventing wrong nondeterminism:

take a nonlinear translation where fusions are replicated:

$$[[u(y).P]]_{u \multimap u'} = u'(y).(y')(!y \multimap y' / [[P]]_{uy \multimap u'y'})$$

and consider

$$\begin{aligned} & [[u(x).\mathbf{0} / v(y).y().\mathbf{0} / \bar{u}z / \bar{v}z / \bar{z}]] \\ &= u(x).(x')!x \multimap x' / v(y).(y')(!y \multimap y' / y'().\mathbf{0}) / \bar{u}z / \bar{v}z / \bar{z} \\ &\rightarrow\rightarrow (x'y')(!z \multimap x' / !z \multimap y' / y'() / \bar{z}) \end{aligned}$$

distributed choice in the linear forwarder calculus

extend the calculus with *choice* as follows:

$$P ::= \dots / x(\tilde{u}).P + y(\tilde{v}).Q$$

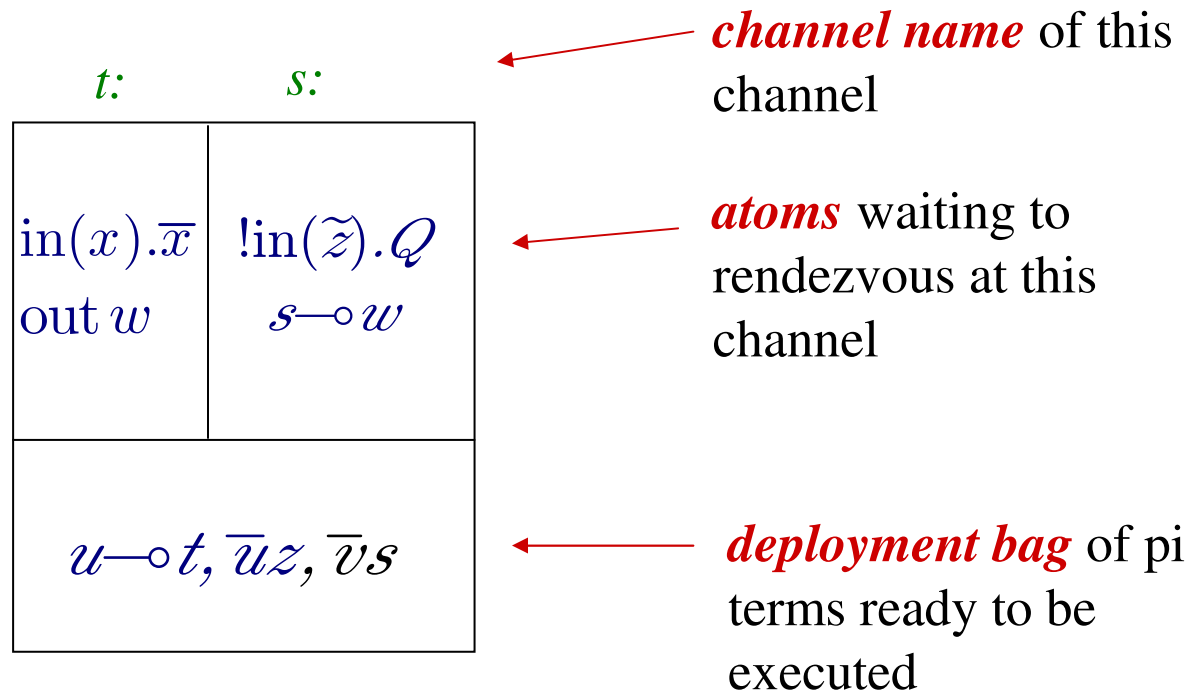
along with the structural congruence law $P+Q \equiv Q+P$ and the reduction rule

$$\bar{x} \tilde{w} / x(\tilde{u}).P + y(\tilde{v}).Q \rightarrow P\{\tilde{w}/\tilde{u}\}$$

the encoding $\llbracket \cdot \rrbracket$ is extended with the additional rule that $\llbracket x(\tilde{u}).P + y(\tilde{v}).Q \rrbracket_{\tilde{z}} =$

$$\begin{array}{ll} x(\tilde{u}).\llbracket P \rrbracket_{\tilde{u}\tilde{z}} + y(\tilde{v}).\llbracket Q \rrbracket_{\tilde{v}\tilde{z}} & \text{if } x \notin \tilde{z}, y \notin \tilde{z} \\ (y')(y \multimap y' / x(\tilde{u}).(y' \multimap y / \llbracket P \rrbracket_{\tilde{u}\tilde{z}}) + y'(\tilde{v}).\llbracket Q \rrbracket_{\tilde{v}\tilde{z}}) & \text{if } x \notin \tilde{z}, y \in \tilde{z} \\ (x')(x \multimap x' / x'(\tilde{u}).\llbracket P \rrbracket_{\tilde{u}\tilde{z}} + y(\tilde{v}).(x' \multimap x / \llbracket Q \rrbracket_{\tilde{v}\tilde{z}})) & \text{if } x \in \tilde{z}, y \notin \tilde{z} \\ (x'y')(x \multimap x' / y \multimap y' / x'(\tilde{u}).(y' \multimap y / \llbracket P \rrbracket_{\tilde{u}\tilde{z}}) + y'(\tilde{v}).(x' \multimap x / \llbracket Q \rrbracket_{\tilde{v}\tilde{z}})) & \text{if } x \in \tilde{z}, y \in \tilde{z} \end{array}$$

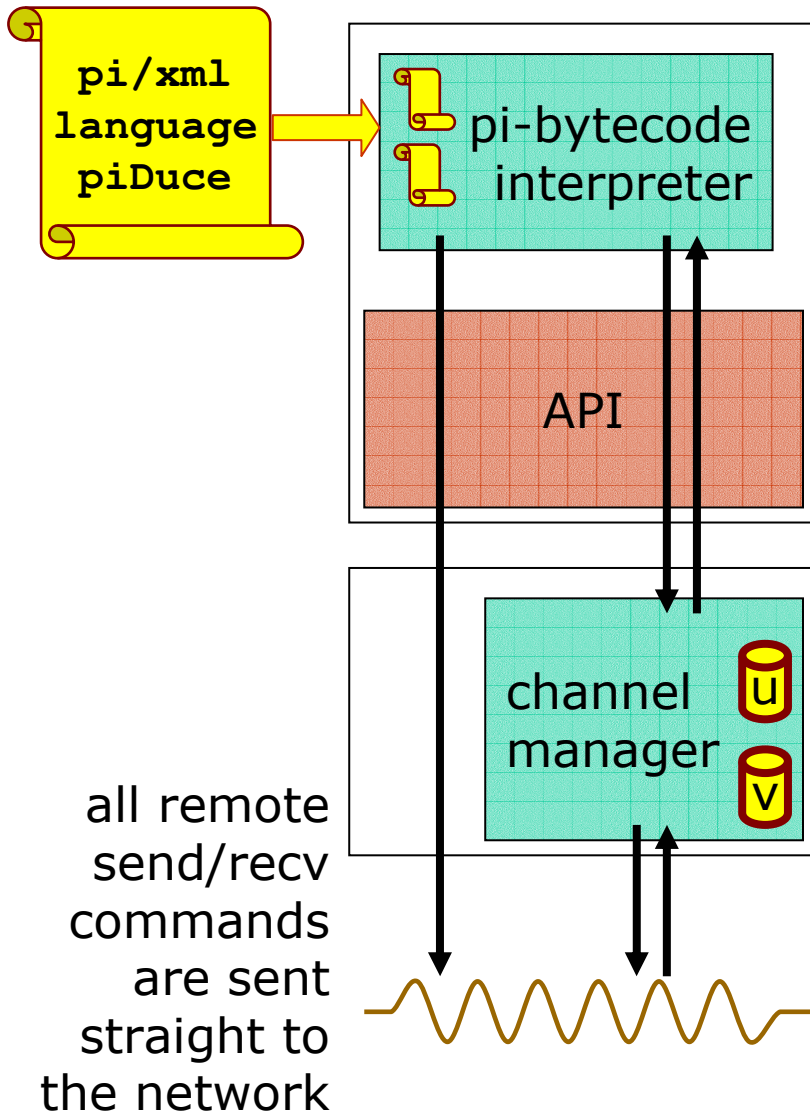
the BoPi machine, logically



- the system is composed *only* of a collection of these distributed channel machines
- this one corresponds to

$$t(x).\bar{x} \mid \bar{t}w \mid !s(z).Q \mid s \multimap u \mid u \multimap t \mid \bar{u}z \mid \bar{v}s$$

the BoPi machine in practice (1)



the interpreter contains a list of thread-closures. `spawn` adds to this list.

```
void send(chan, E, cont);  
void asend(chan, E);  
void rcv(pattern, cont);
```

each drum is a list of blocked tuples:

```
SND(u, E, continuation)  
RCV(pattern, continuation)  
RCV(pattern, forwarding-addr)
```

tuples arrive off the network.
if this lets a drum complete a SND/RCV pair, then they unblock and their continuations are invoked.

the BoPi machine in practice (2): piDuce

- *pi calculus is extended with native XML data types*
- *inputs are extended with pattern matching a la ML*

$V ::=$	value		
	ℓ		
	x	$P ::=$	process
	u		0
	0		$x(T).P$
	$a[V].V$		P/P
$T ::=$	pattern		$\bar{x}[V]$
	0		$(x)P$
	$a[T].T$		X
	T/T		$\text{rec } X. P$
	u		

the BoPi machine in practice (3): piDuce semantics

- the *pattern matching* of a value V with respect to a pattern T , written $V \in T \Rightarrow \sigma$, is the least relation closed under structural congruence and satisfying the rules:

$$\mathbf{0} \in \mathbf{0} \Rightarrow \emptyset \quad V \in u \Rightarrow [u \mapsto V] \quad \frac{V \in T \Rightarrow \sigma \quad V' \in T' \Rightarrow \sigma'}{a[V].V' \in a[T].T' \Rightarrow \sigma + \sigma'}$$

$$\frac{a[V] \in a[T] \Rightarrow \sigma \quad V' \in T' / T'' \Rightarrow \sigma'}{a[V].V' \in a[T].T' / T'' \Rightarrow \sigma + \sigma'}$$

- the *reduction relation*, noted \rightarrow , is the least relation closed under structural congruence, “- / -” and (x) -, and satisfying the rule:

$$\frac{V \in T \Rightarrow \sigma}{\bar{x}[V] / x(T).P \rightarrow P\sigma}$$

what we are doing/discovering

- *linear forwarders makes for **easy implementation** and **easy and strong proofs of correctness***
- *fusions/linear forwarders allow for **optimisations** at source level, by “pre-deploying” fragments to their expected destination*
“on-the-fly deployments” may be performed in parallel with other operations
- *the distributed machine has been written in Java: substantial work is needed to build a **language** on top of it*
 - **XML data types** (in the next release – July 2004)
 - **transactions and rollbacks like Xlang/BPEL**

fusion machine & join

fm & join: have a unique channel manager for every channel

- *inputs and outputs are deployed to the corresponding manager*
- *reactions occur locally*

fm \neq join:

- **fm granularity is finer:** it is possible to locate one channel per machine, in join all co-defined channels are on the same machine
- **fm channel behaviours may be re-programmed:** in join the channel behaviour is fixed once for all (a received name cannot be redefined)