New directions in implementing the pi calculus

the fusion project at bologna

Lucian Wischik

CaberNet Radicals October 2002

Do you know about the pi calculus? …

… Bill Gates does

the pi calculus

$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello'', \text{“}world''\rangle$ // get it to relay this message
$)$

```
void mbox(const char *text, const char *caption)
{ MessageBox(hwnd,text,caption,MB_OK); }
```

Q. How does it work? ✍

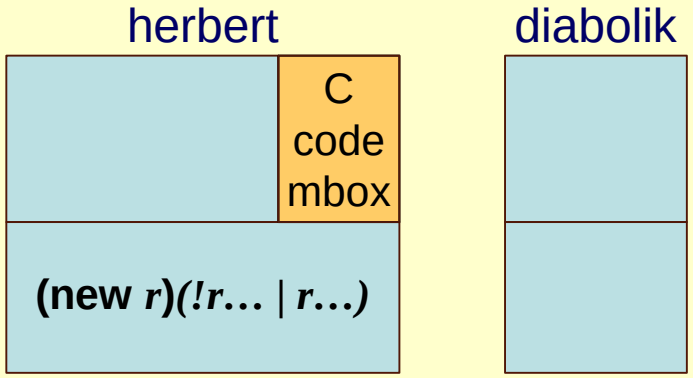
Q. What's it good for? ✍

Q. How to distribute it? ✍

Q. A behavioural type system? ...
Transactions? Compensations?

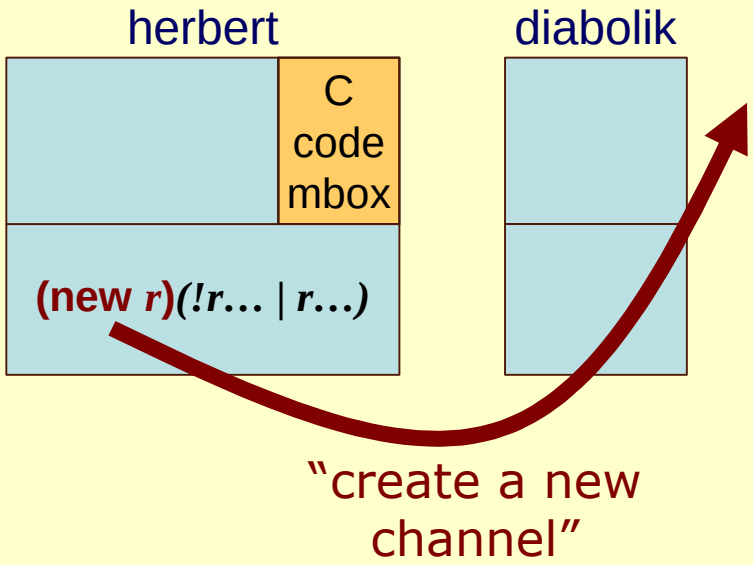
distributed channel machine

$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello\text{”}, \text{“}world\text{”}\rangle$ // get it to relay this message
$)$



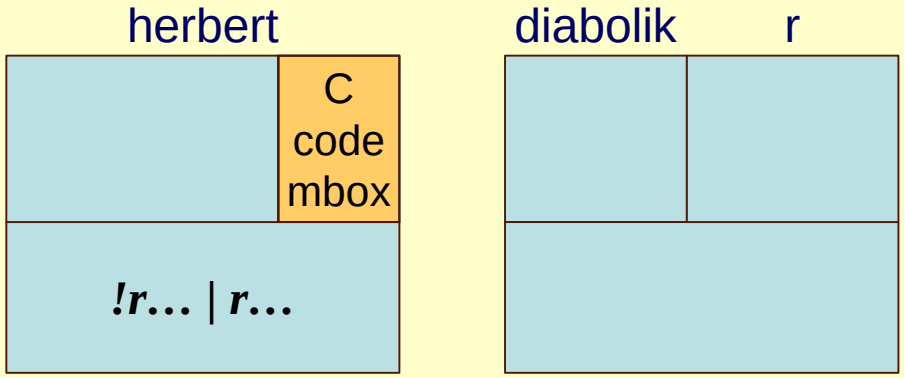
distributed channel machine

$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello\text{”}, \text{“}world\text{”}\rangle$ // get it to relay this message
$)$



distributed channel machine

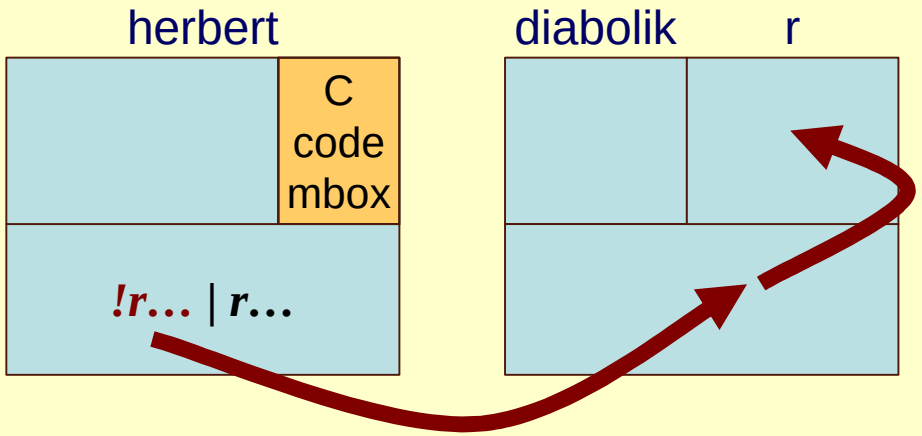
$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello\text{”}, \text{“}world\text{”}\rangle$ // get it to relay this message
$)$



Optimisation: channel creation could be asynchronous (not requiring network messages) if herbert can generate a GUID for diabolik

distributed channel machine

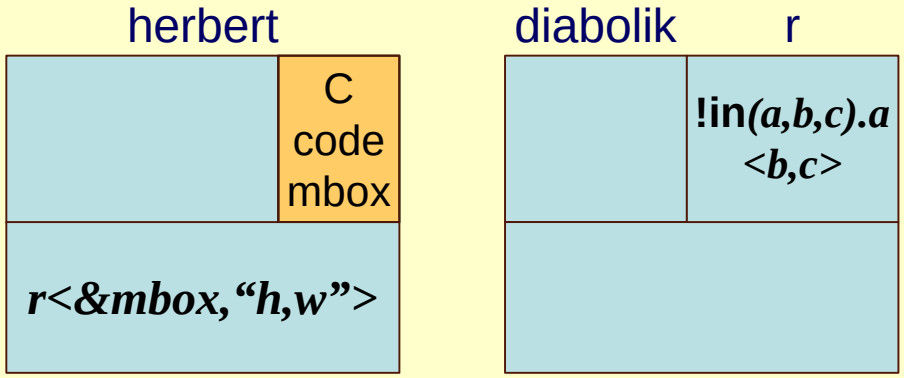
$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello\text{”}, \text{“}world\text{”}\rangle$ // get it to relay this message
$)$



Implementation note: a server thread accepts migrants and places them in the lower half; a worker thread picks up jobs from there, and executes them

distributed channel machine

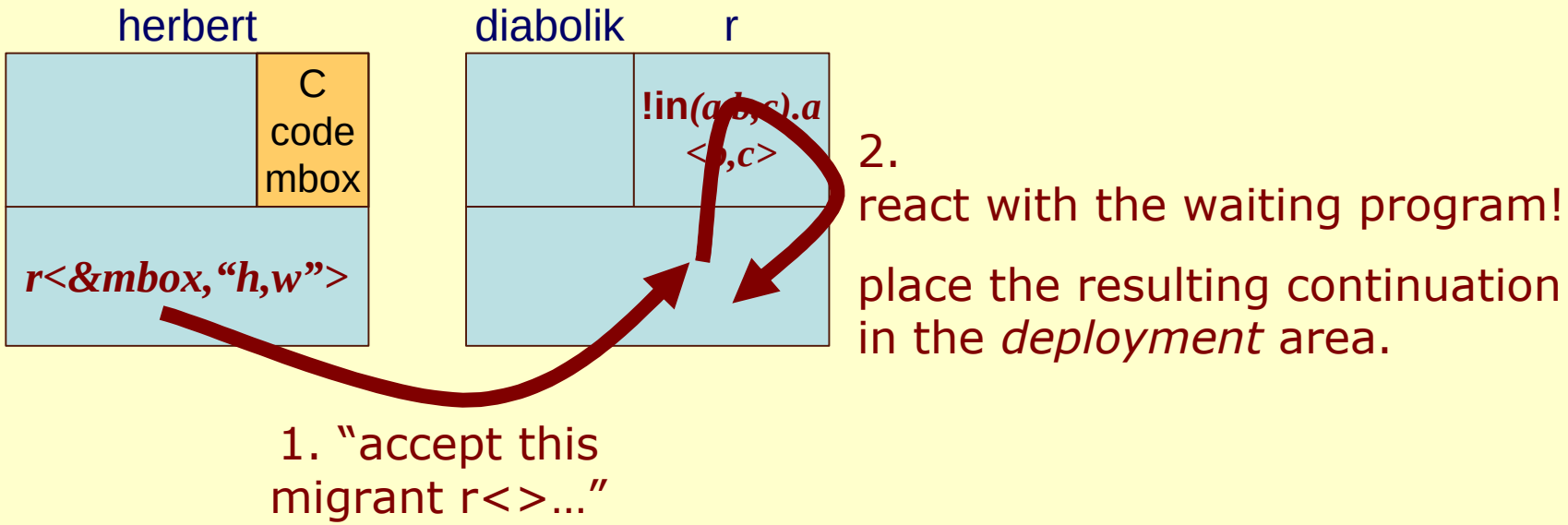
$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello\text{”}, \text{“}world\text{”}\rangle$ // get it to relay this message
$)$



Tech note: the address &mbox is a ‘network address’ comprising herbert’s IP number, port number, and the function’s location in memory.

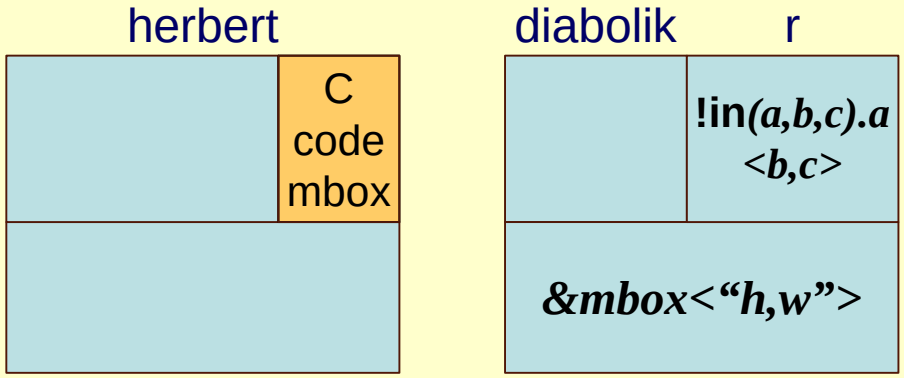
distributed channel machine

$$
\begin{array}{ll}
(\text{new } r@diabolik.unibo.it)(& \text{// create a fresh channel at this location} \\
\quad !r(a,b,c).\overline{a}\langle b,c\rangle & \text{// place a relaying server there} \\
\quad \mid \overline{r}\langle \&mbox, \text{“}hello\text{”}, \text{“}world\text{”}\rangle & \text{// get it to relay this message} \\
) &
\end{array}
$$



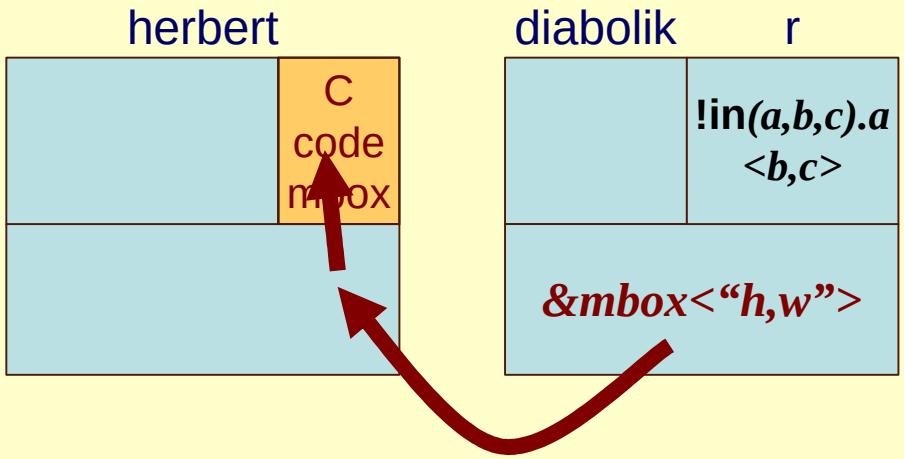
distributed channel machine

$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello\text{”}, \text{“}world\text{”}\rangle$ // get it to relay this message
$)$



distributed channel machine

$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello\text{”}, \text{“}world\text{”}\rangle$ // get it to relay this message
$)$

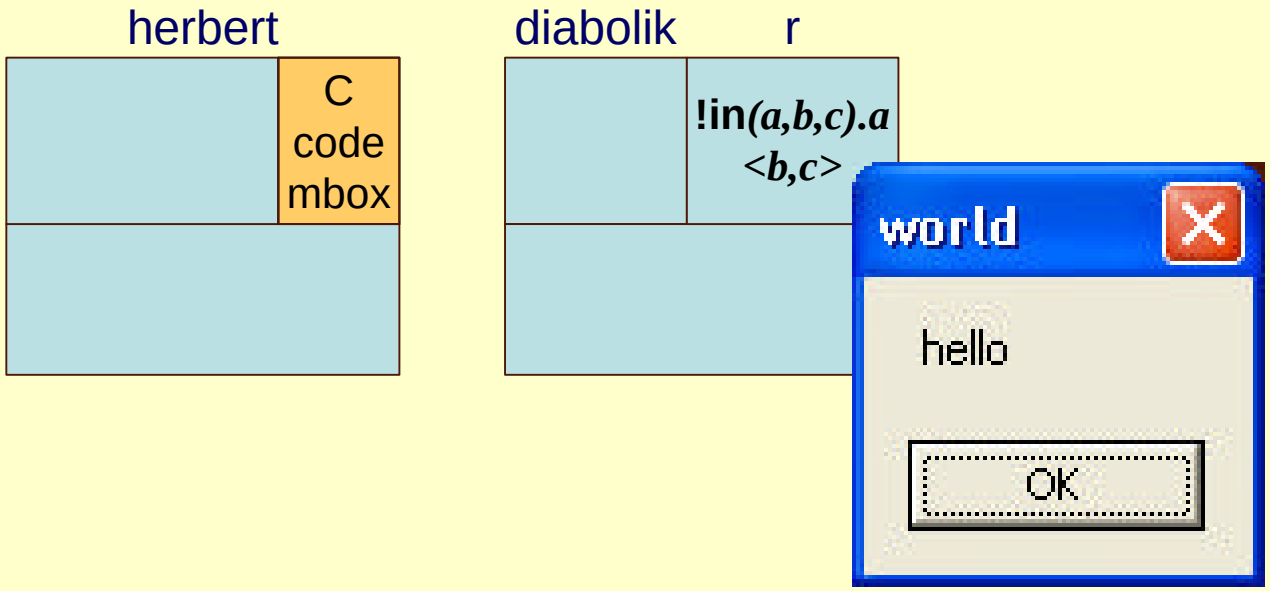


(again, it reacts immediately,
this time by invoking C code)

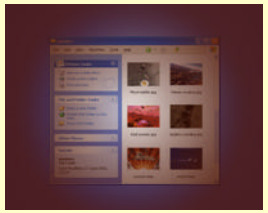
Pi calculus feature:
integrates neatly with
conventional langs…
invoking a function
= sending a message.

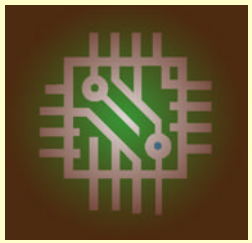
distributed channel machine

$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello\text{”}, \text{“}world\text{”}\rangle$ // get it to relay this message
$)$



vertical pi market

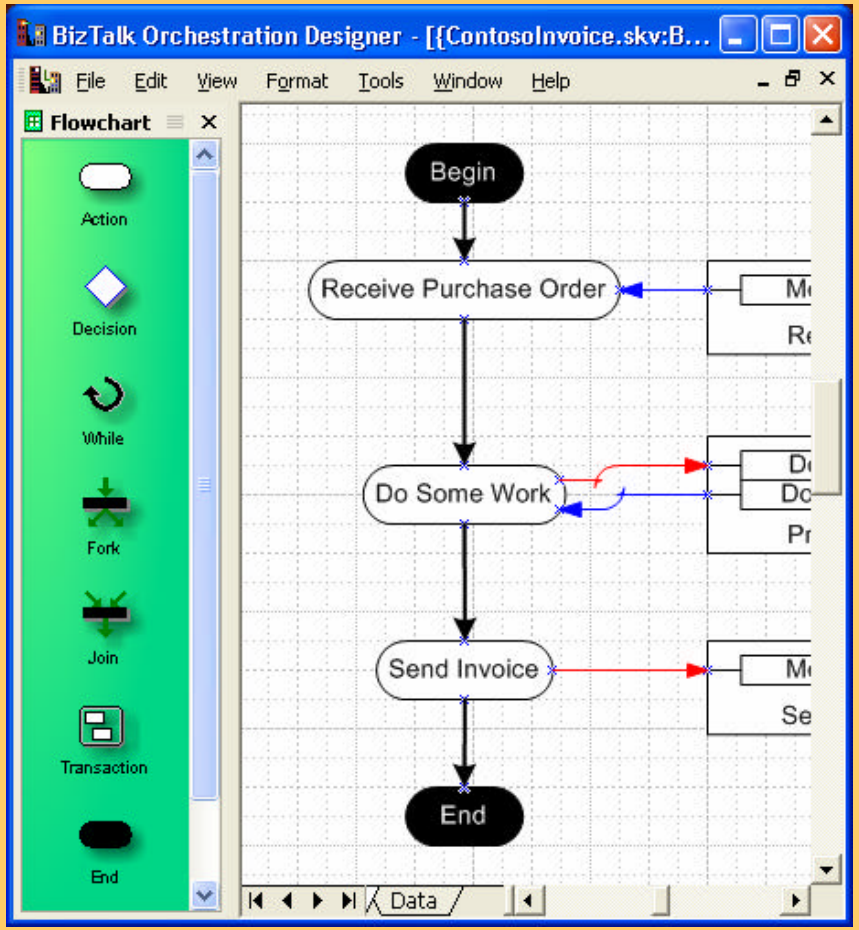




Business to Business:

- long running transactions (weeks or months)
- messaging is reliable and audited
- high latency
- trust barriers
- upon a crash, boot up hard disk on another machine and completely restore system state

Microsoft Biztalk, 2000-2002

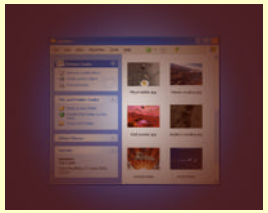


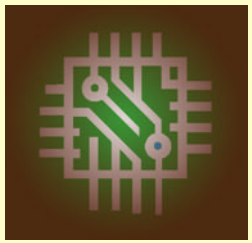
vertical pi market

Business to Business:

Internet, Wireless

- behavioural type system to enforce protocols
- high latency
- messaging may be reliable or unreliable
- built-in XML datatypes



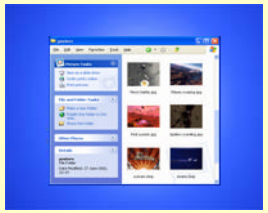


Microsoft Highwire

```
schedule Eg (order,invoice: Biz)
{
   new (dbg: Debug,  done: Completion)

   sequence
   {
      order [(dat^: OrderData) {}];
      dbg [(WriteLine, “started!”) {}];

      parallel
      {
         call Worker1 (^dat.a, done, 1);
         call Worker2 (^dat.b, done, 2);
      }

      done [(~1, ~2) {}];
      dbg [(WriteLine, “work fini”) {}];
      invoice [(^dat.c) {}];

   }

}
```

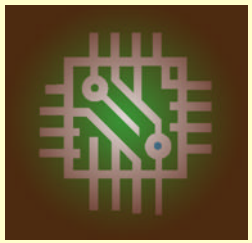
vertical pi market

Internet, Wireless

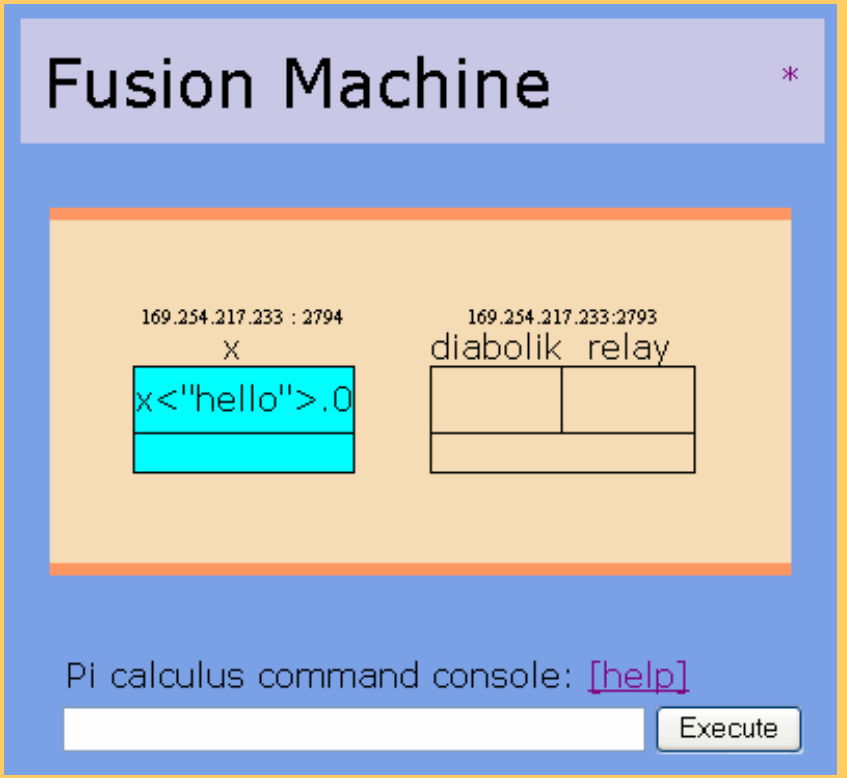


Desktop apps

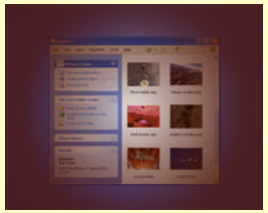
- interop with normal languages
- no failures
- a way to write multi-threading

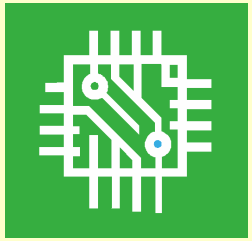


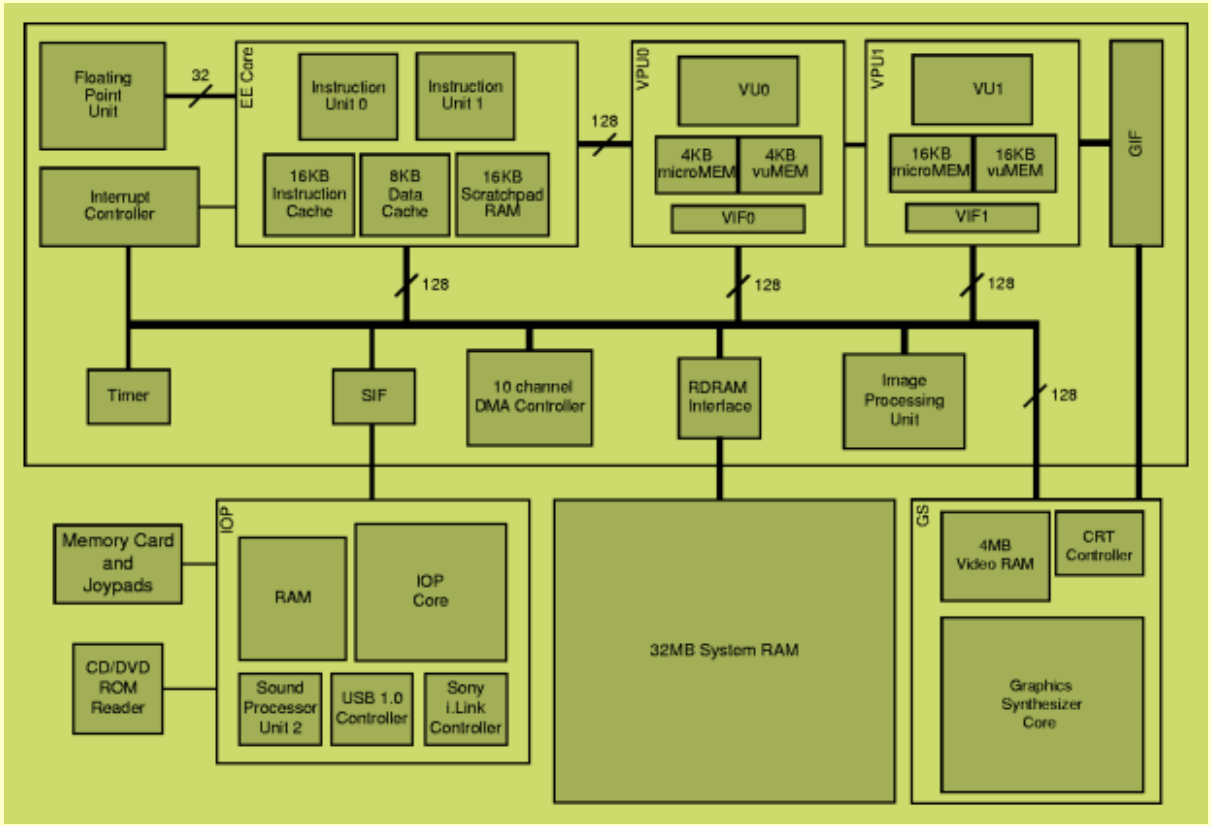
Fusion Machine at Bologna



vertical pi market



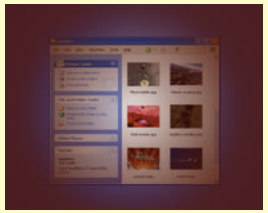


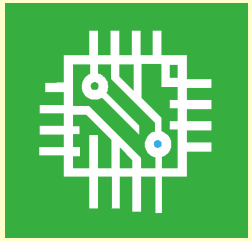


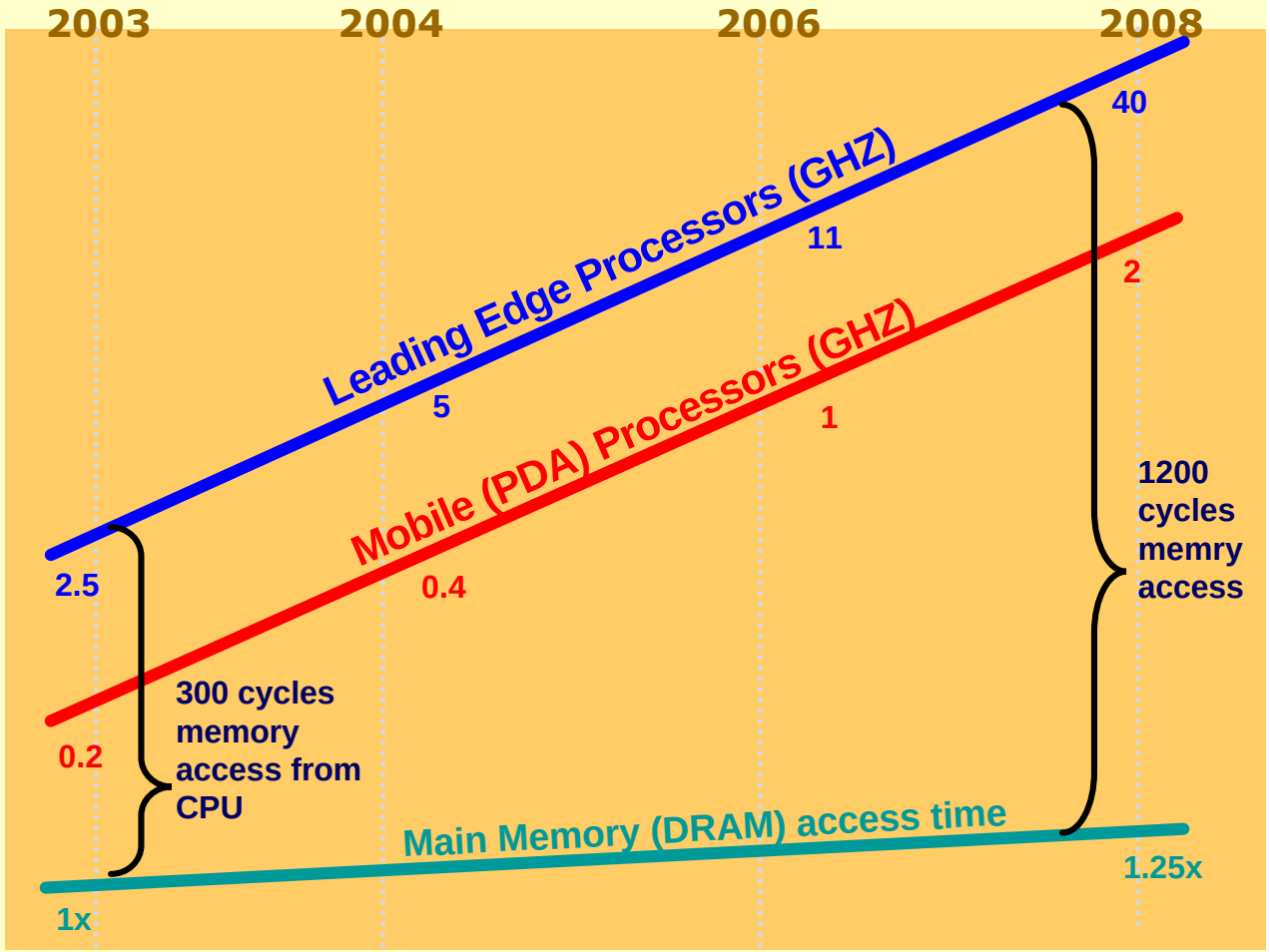
Computer internals

- heteregenous architectures – special-purpose chips, connected in odd ways
- e.g. Playstation 2

vertical pi market







Computer internals

- high latency – a m'board is a distributed system!
- failures?

vertical pi market

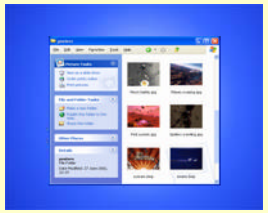
Biztalk

Highwire

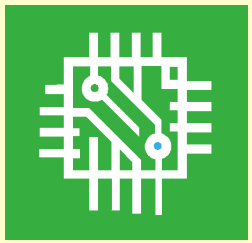
mob. agents

Fusion machine

Jocaml



Pict



???

Common features

- High parallelism
- High latency
- Heterogenous

Common solutions

- Messaging
- Rendezvous

Q. How to write distribution in the pi calculus?

Q. How to treat restriction?

the baddies

nomadic pict, distributed pi, ambients

"The pi calculus has no notion of location. Therefore we add locations to the calculus."

$$l[\,\text{migrate}\,m \mid \overline{u}\,x.P\,] \quad m[\,u(y).Q\,]$$

"Add commands to the language like ***migrate*** and ***newloc***."

$$\text{newloc}\,m$$
$$\text{new}\,x.P$$

"We get a calculus that's good for writing mobile agents."

TO WRITE DISTRIBUTION IN THE PI CALCULUS

the goodies

fusion machine, highwire, biztalk

"The pi calculus is already a calculus about locations: each channel-name is a location."

$$\overline{m}x.P \quad m(y).Q$$

"Reuse the ***new*** command; let it also choose physical adjacency."

$$\text{new}\,y@m.P$$

"We get a calculus that's easy to use and prove."

the baddies
theoreticians

"***Restriction*** declares a name hidden: no one else can use it."

$$(x)P$$

$$\overline{x} \mid (x)x.Q$$

"It allows for alpha-renaming and scope extrusion."

$$P \mid (x)Q \rightleftharpoons (x)(P \mid Q)$$

$$(x)P \rightleftharpoons (y)P\{y/x\}$$

TO HANDLE RESTRICTION IN THE PI CALCULUS

the goodies
implementors

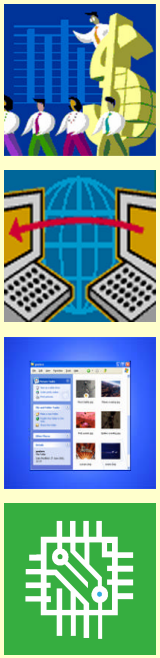
"***Restriction*** is a command which creates a fresh channel."

$$\text{new}\, x.P \rightharpoonup P\{^{1.2.5.1}/_{x}\}$$

$$\overline{1.2.5.1} \mid 1.2.5.1.Q$$

"Alpha-renaming and scope extrusion are not implementable."

Theorem: two different operational semantics, but amount to the same thing



pi calculus

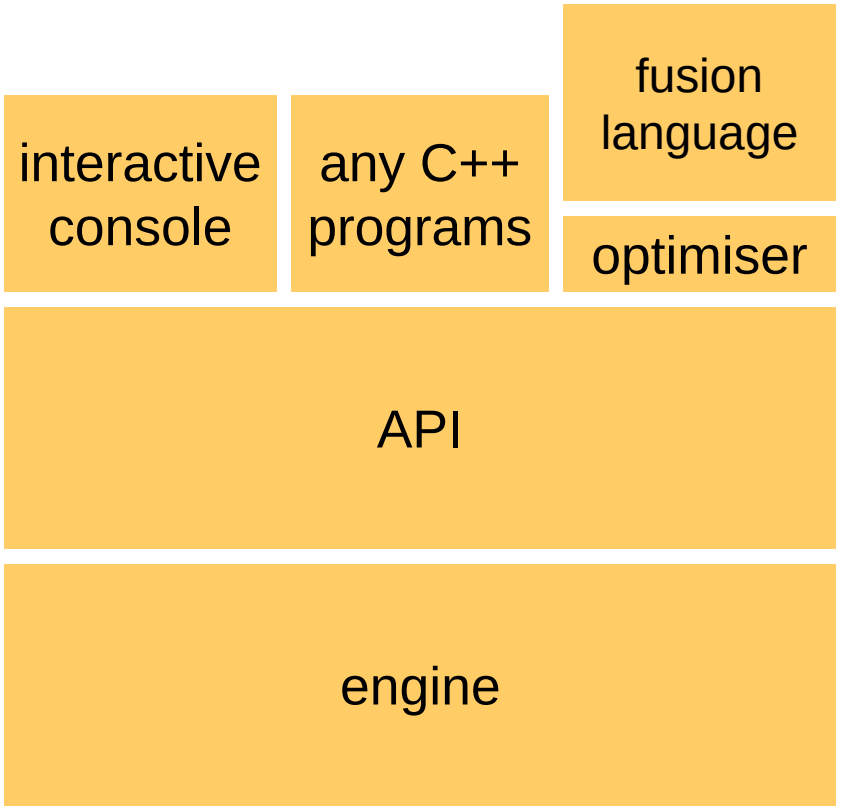
fusion machine
highwire, biztalk

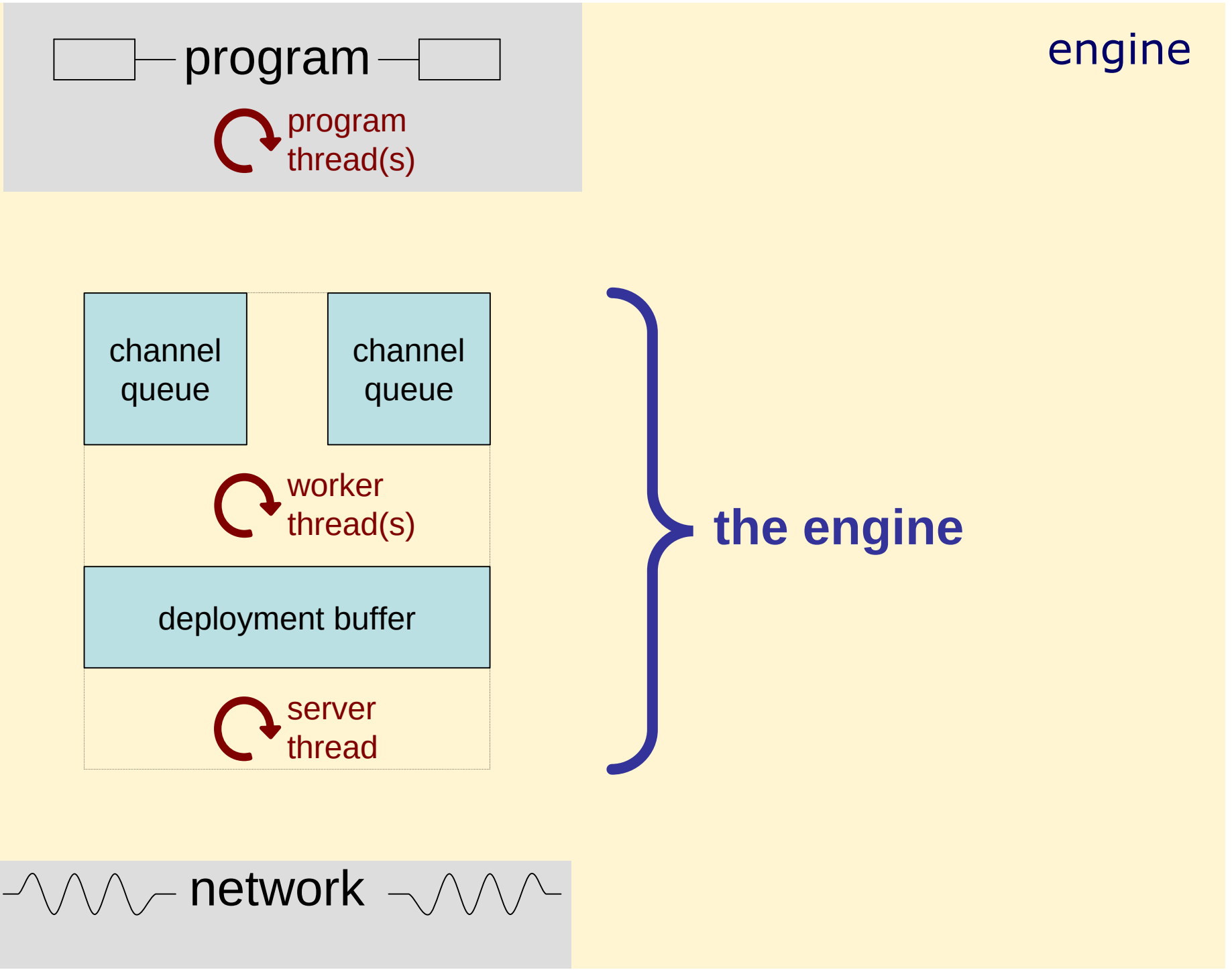
$\text{new}\, y@m.P$

$\text{new}\, x.P \rightharpoonup \ldots$

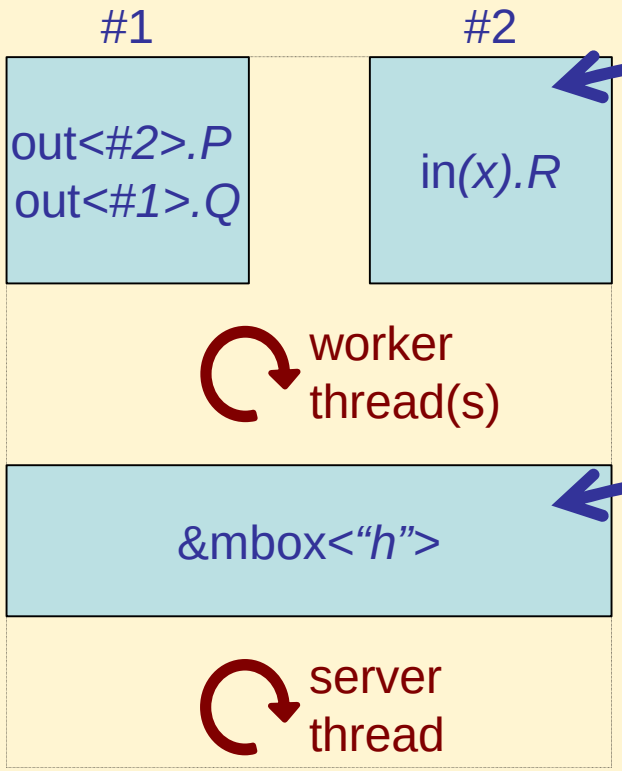
The challenge it is to design a language easy enough
that ordinary working programmers can use it.

supplemental slides


interactive console
any C++ programs
fusion language
optimiser
API
engine

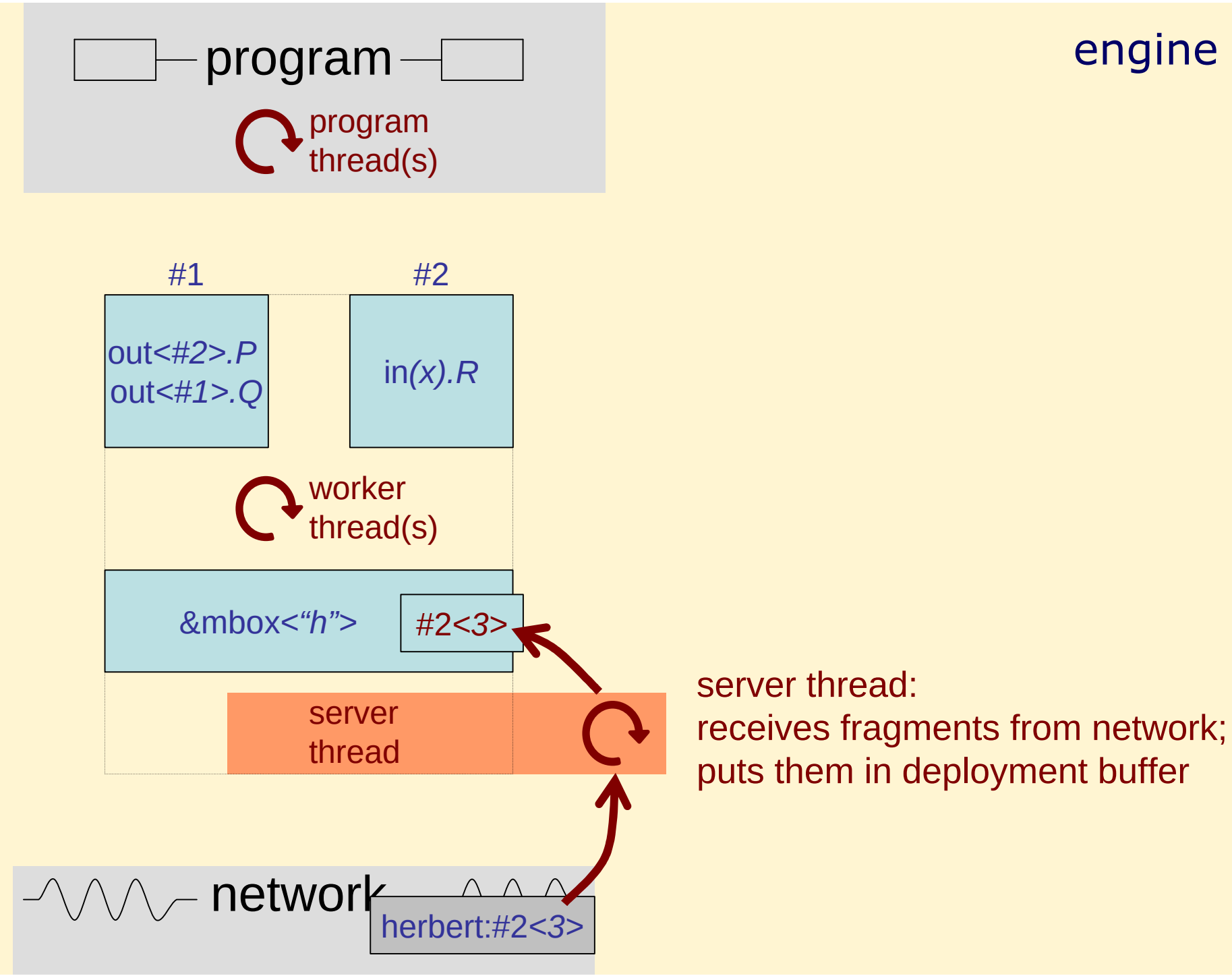

program
program
thread(s)
engine
channel
queue
channel
queue
worker
thread(s)
deployment buffer
server
thread
the engine
network

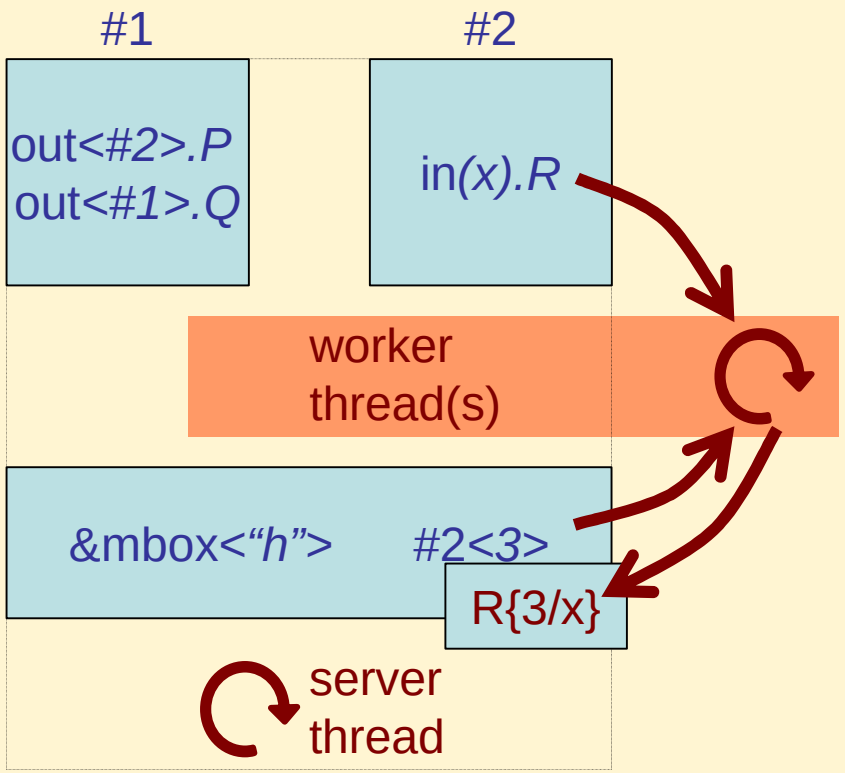
engine



channel queue:
contains program fragments
waiting to react
at channel#2 here

deployment buffer:
contains program fragments
ready to be executed/migrated
right now

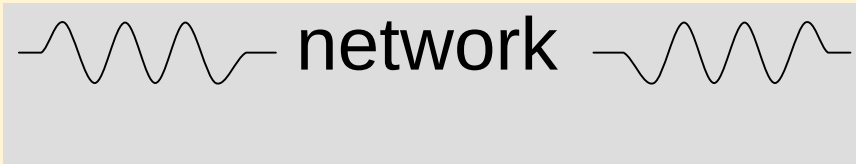

program
program
thread(s)
engine
#1
#2
out<#2>.P
out<#1>.Q
in(x).R
worker
thread(s)
&mbox<"h">
#2<3>
server
thread
server thread:
receives fragments from network;
puts them in deployment buffer
network
herbert:#2<3>

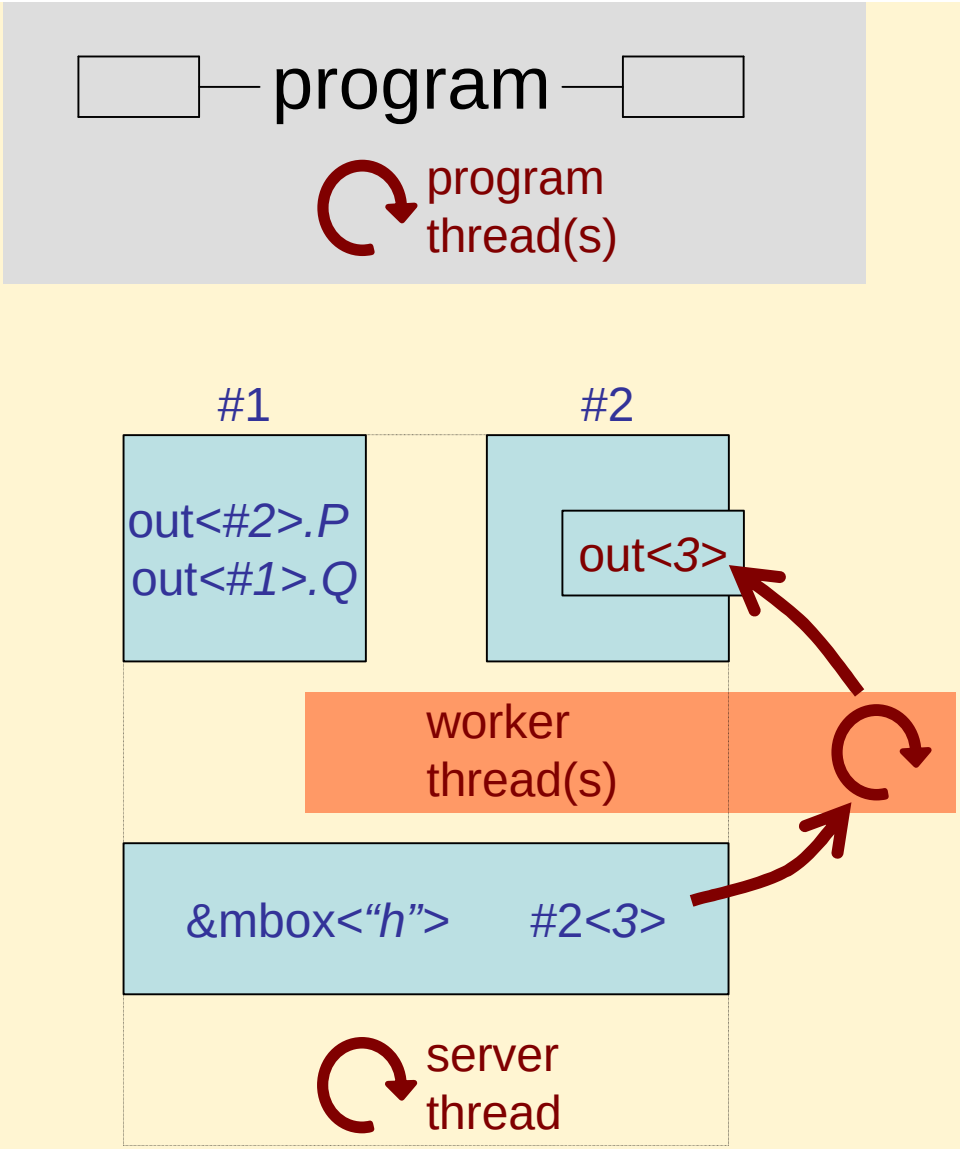


worker thread:

takes fragment from deployment buffer;

1. maybe can react it immediately, in which case puts continuation back in the deployment buffer

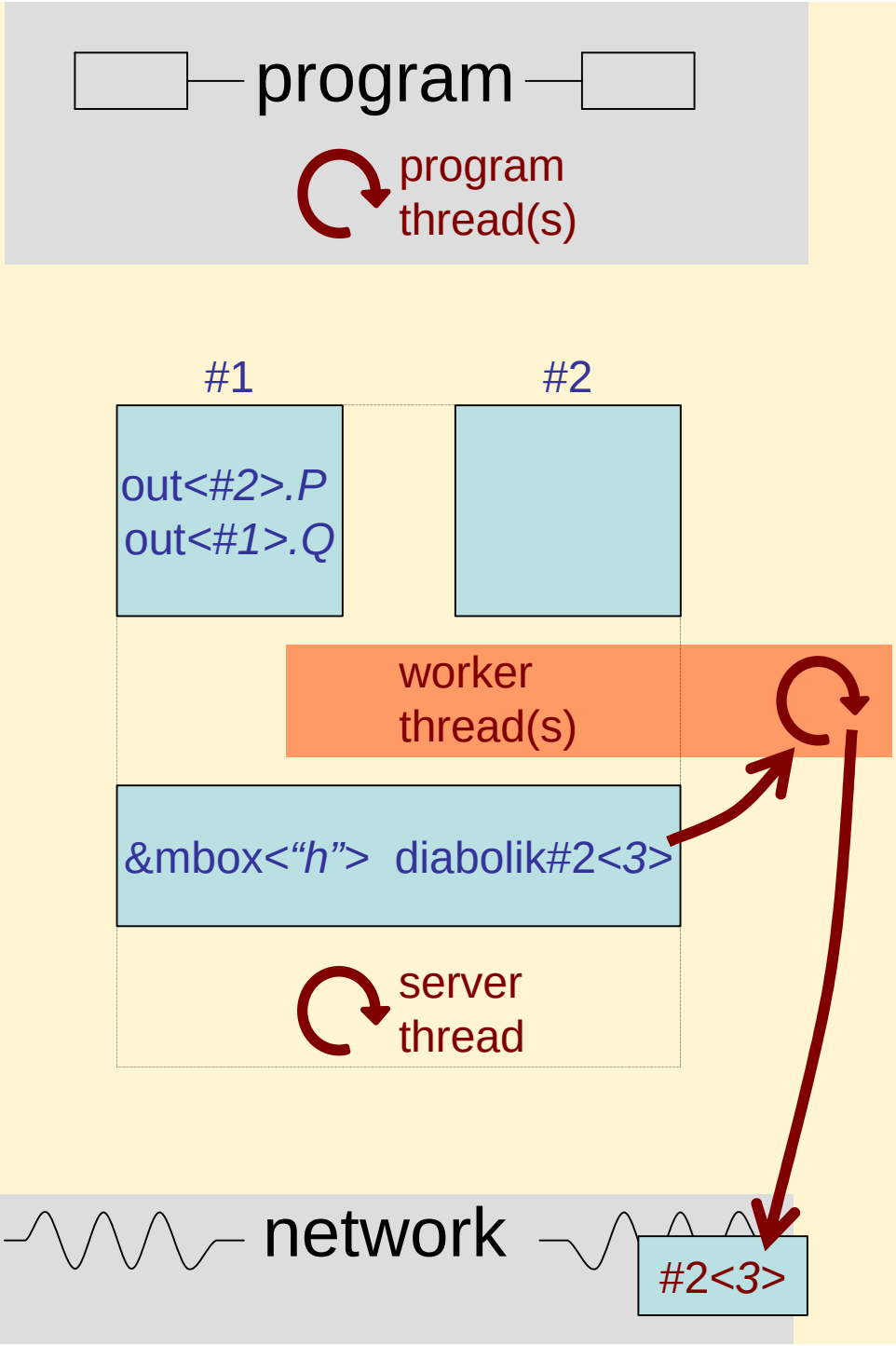




worker thread:

takes fragment from deployment buffer;

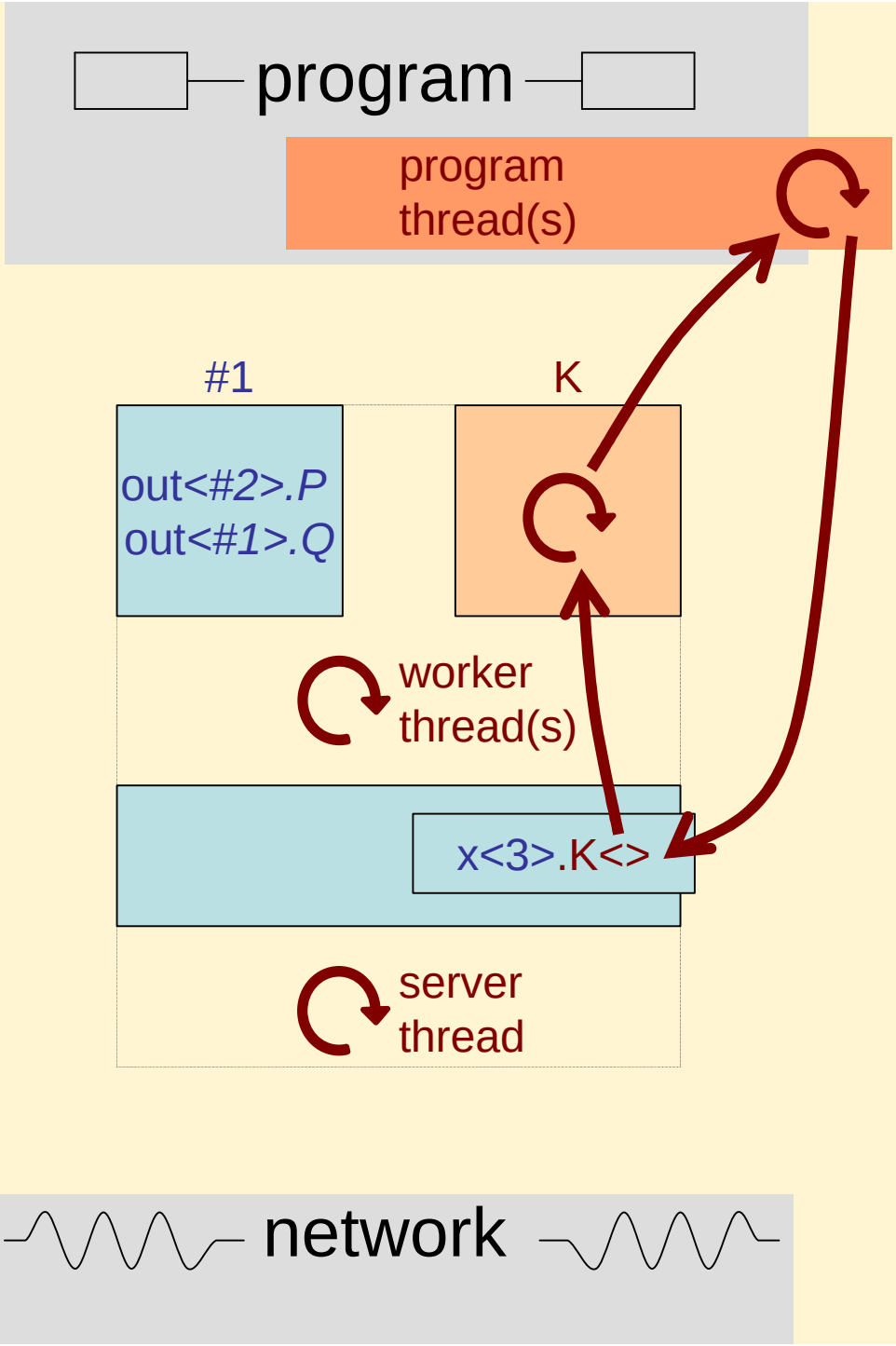
2. maybe it can’t react yet, in which case puts fragment in the channel queue



worker thread:

takes fragment from deployment buffer;

3. maybe it's for a different IP, in which case sends it over the network



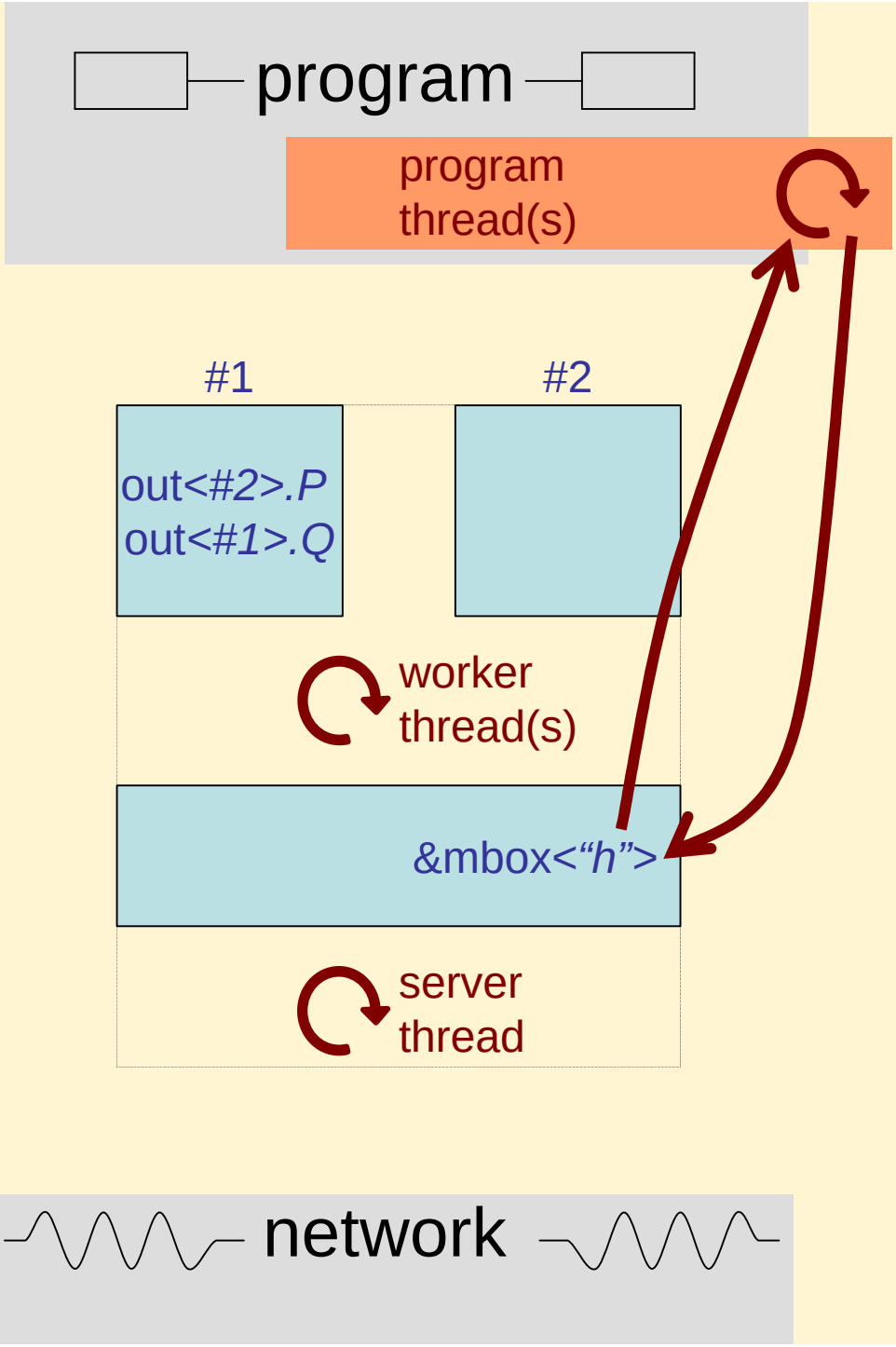
engine

program thread:

can send, receive, create new names, …

for blocking operations, the thread is stored in a dummy *continuation-channel* K:
when the operation completes, this channel is invoked, so waking up the thread

result: a way to integrate C++ code with blocking pi calculus



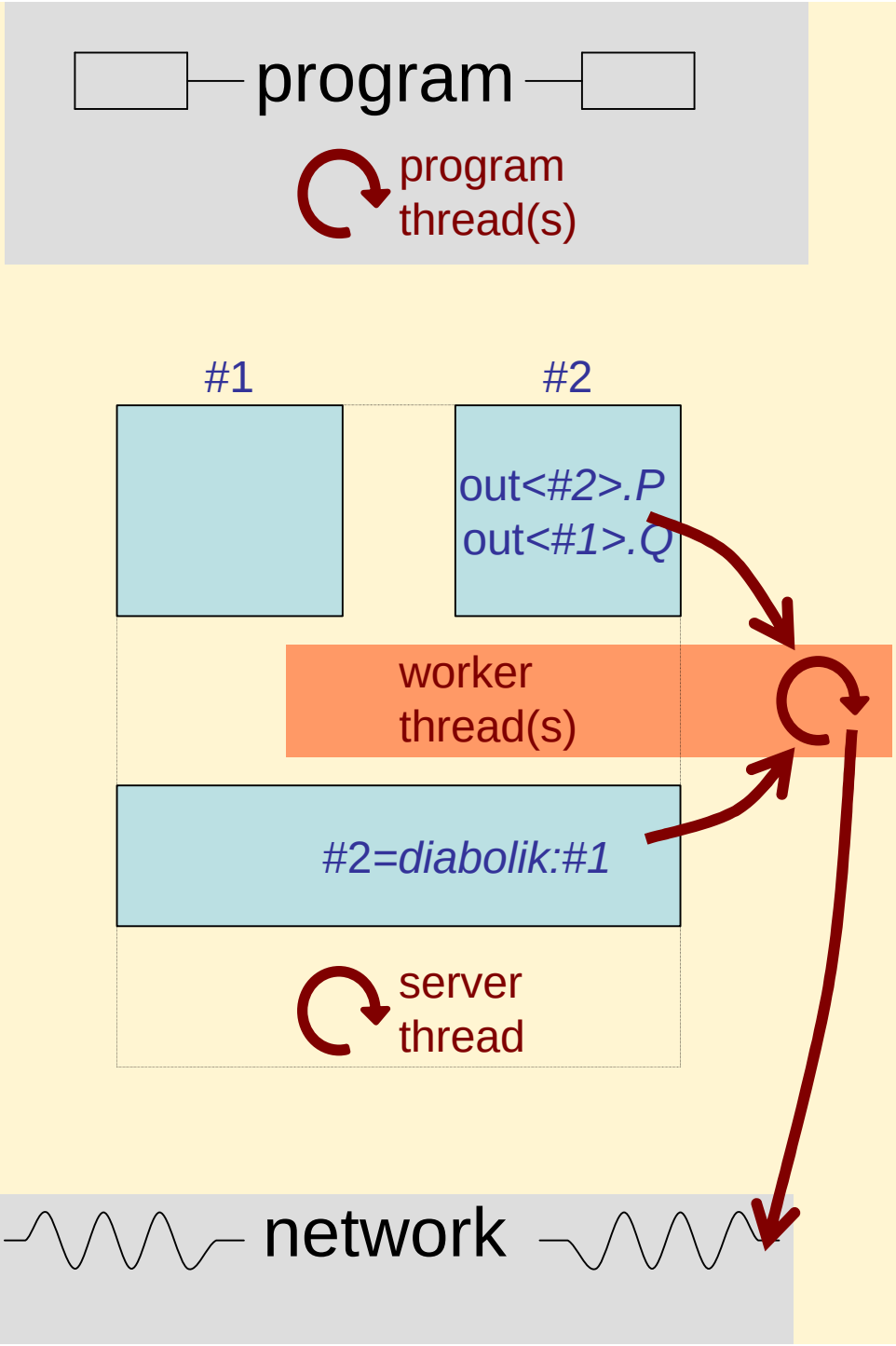
engine

program thread:

frequently "polls" the engine (currently: a message is sent to the main UI thread)

functions like *mbox* can be executed in that thread's context

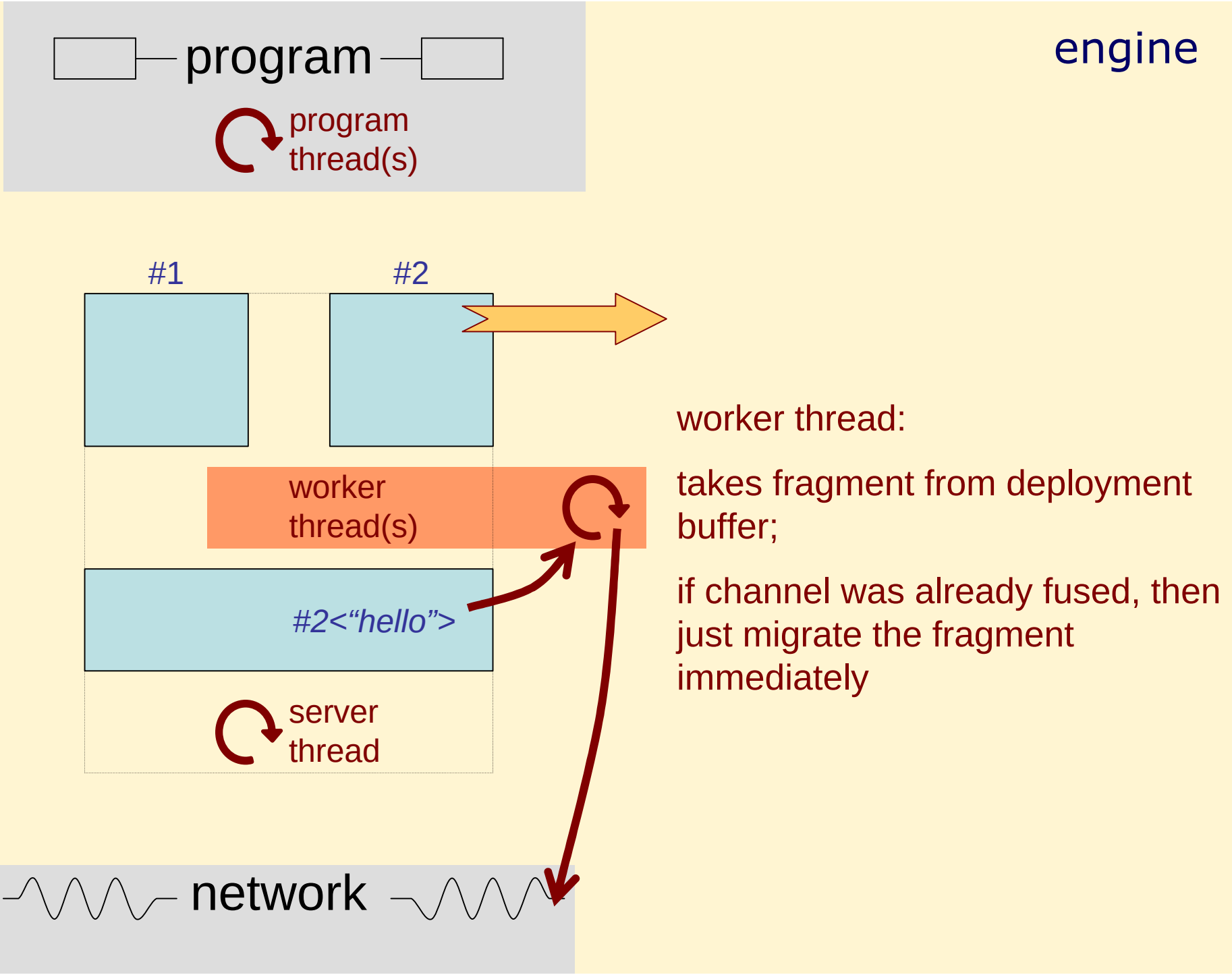
result: ease of use for programmers, who do not have to bother making *mbox* thread-safe



worker thread:

takes fragment from deployment buffer;

if it was a fusion, migrate all the existing contents, and set up a fusion pointer.



worker thread:

takes fragment from deployment buffer;

if channel was already fused, then just migrate the fragment immediately

bytecode

$(\text{new } r@diabolik.unibo.it)($ // create a fresh channel at this location
$\quad !r(a,b,c).\overline{a}\langle b,c\rangle$ // place a relaying server there
$\quad \mid \overline{r}\langle \&mbox, \text{“}hello'', \text{“}world''\rangle$ // get it to relay this message
$)$

bytecode

```
 0  new @1
10  par +40
20  rcv!5(3)
30  snd 6<7,8>
40  nil
50  snd 5<2,3,4>
60  nil
```

stack

```
1  127.9.3.15 : 2793
2  127.9.3.15 : 2793 : 0x400A23
3  “hello”
4  “world”
5
6
7
8
```

IP+port address of the engine running at diabolik

IP+port address of the engine running at diabolik

r will go here at line *0*

a,*b*,*c* will go here at line *20*

Treat functions as addresses

- a name *n* = 2.3.1.7 : 9 : 0x04367110
- so that **`snd(n)`** will invoke the function at that address

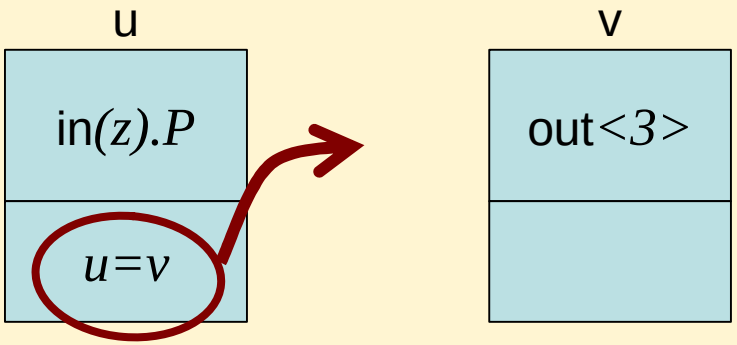
Calling snd/rcv directly from C++

```
{ …          // there's an implicit continuation K after the rcv,
  rcv(x);    // so we stall the thread and put x.K in the work bag.
  …          // When K is invoked, it signals the thread to wake up
}
```

Calling arbitrary pi code from C++

```
pi("u!x.v!y | Q");
pi("u!x."+fun_as_chan(&test2)+"|Q");

void test2()
{ … }
```

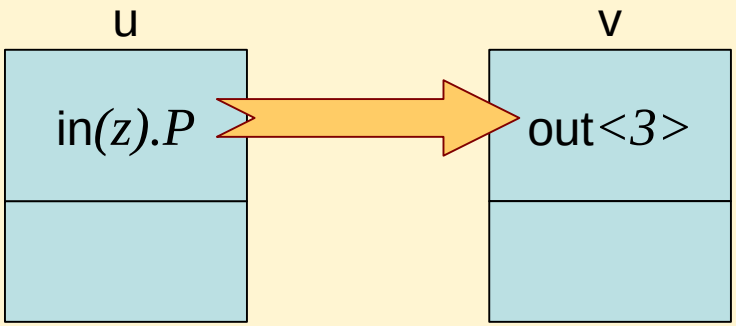
fusions: implementation



A fusion is something that allows two names to be used interchangeably

- We implement with *forwarders*:
- when the fusion $u=v$ is executed, it sets up a forwarder from u to v
- hence: no matter which of u or v you send a message to, the end result is the same.

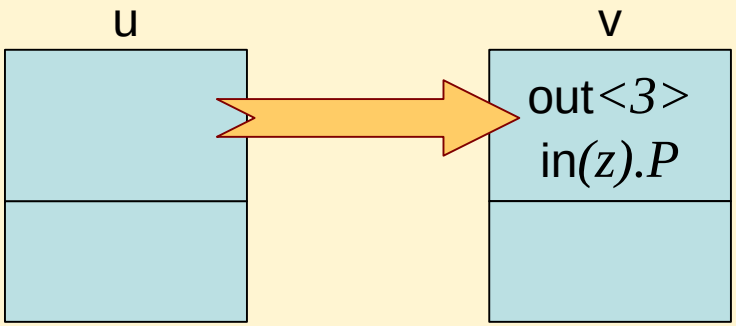
fusions: implementation



A fusion is something that allows two names to be used interchangeably

- We implement with *forwarders*:
- when the fusion $u{=}v$ is executed, it sets up a forwarder from u to v
- hence: no matter which of u or v you send a message to, the end result is the same.

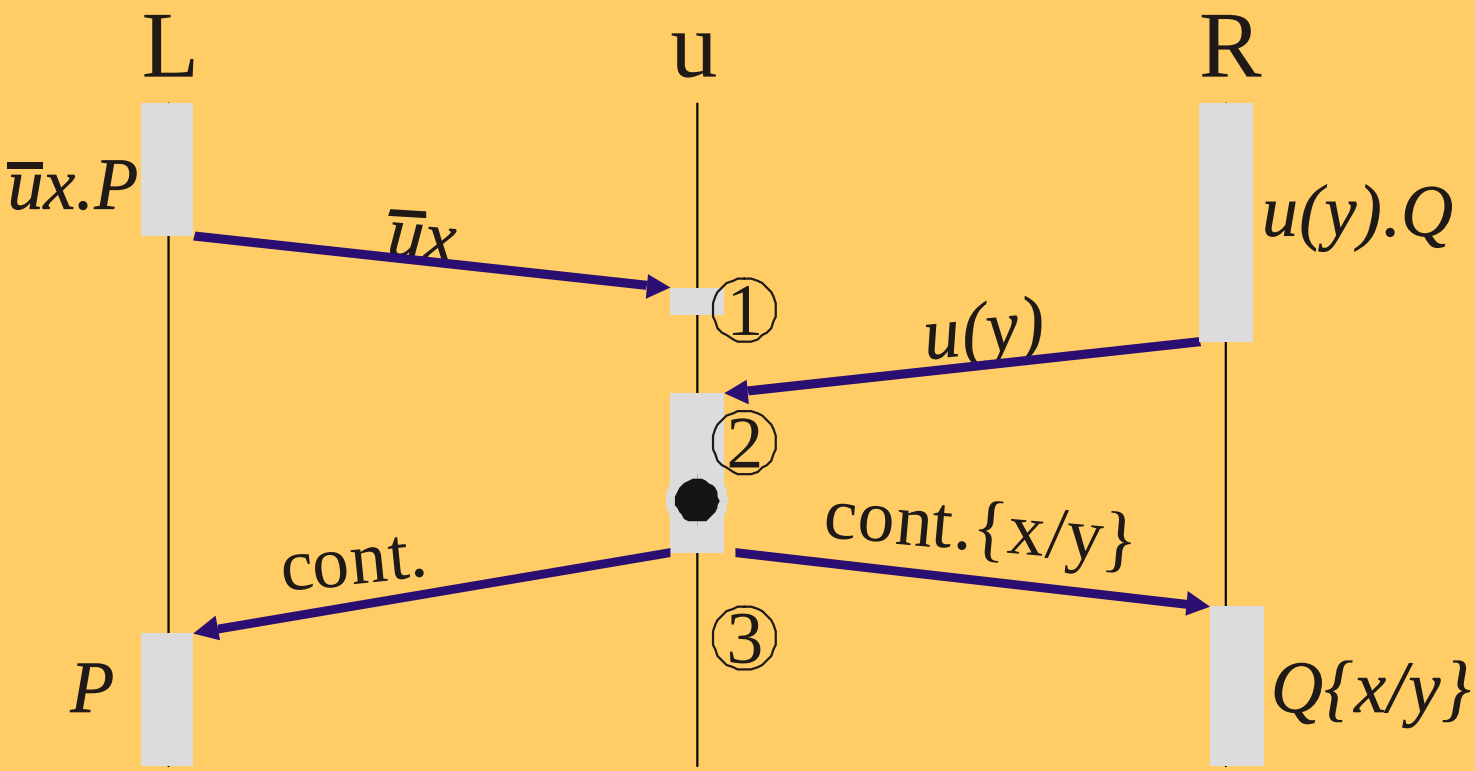
fusions: implementation



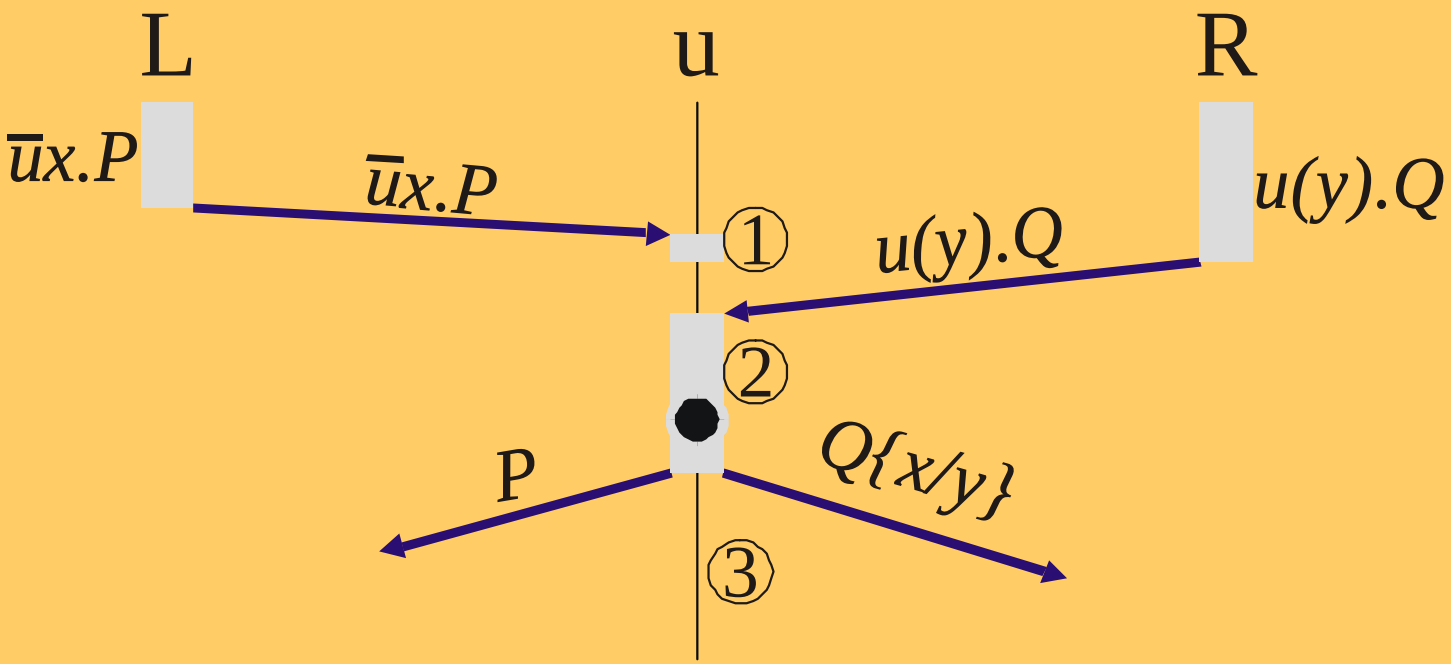
A fusion is something that allows two names to be used interchangeably

- We implement with *forwarders*:
- when the fusion $u{=}v$ is executed, it sets up a forwarder from u to v
- hence: no matter which of u or v you send a message to, the end result is the same.

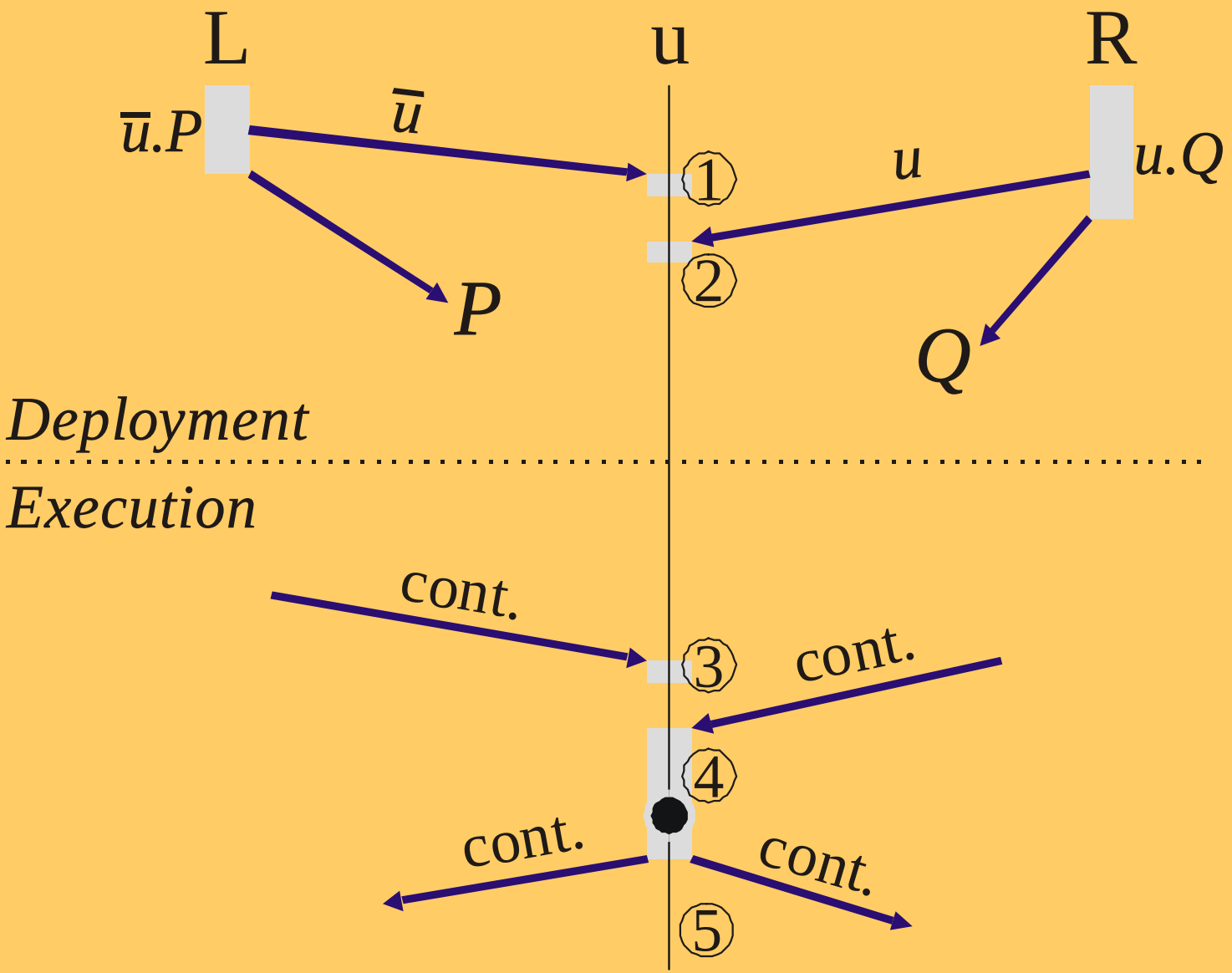
the rendezvous problem



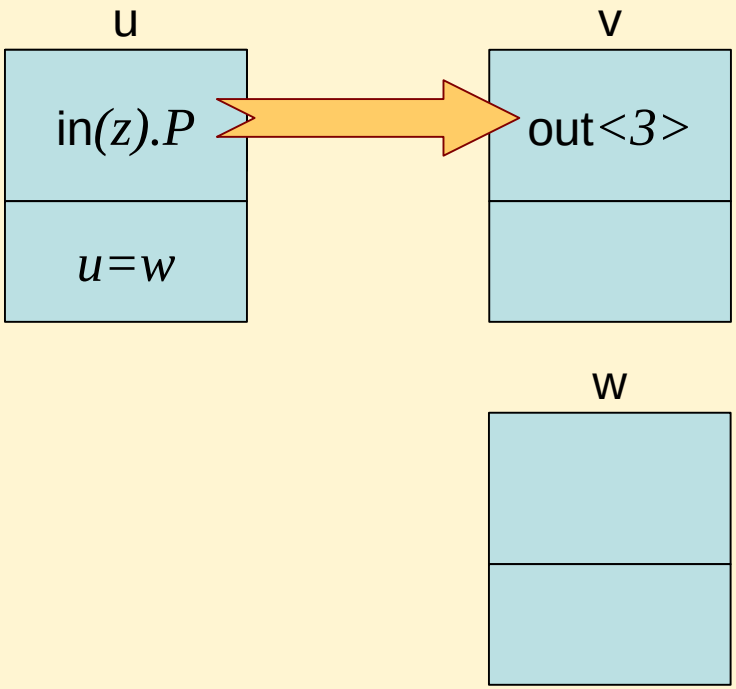
the rendezvous problem



the rendezvous problem



the fusion clash problem

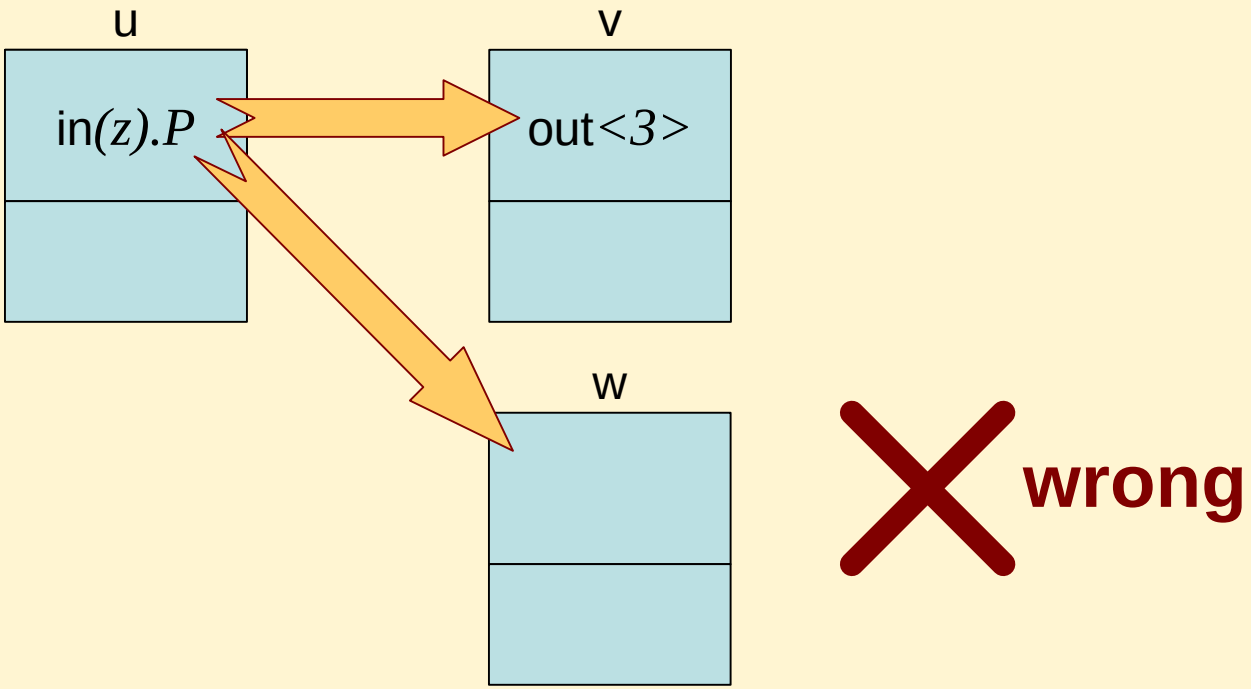


The fusion clash problem:

- If u becomes fused to two different names, how does it know whether to send its code to v or w?

- answer: migrate the fusion down the forwarders until it can be fulfilled; forwarders respect a total order on names.

the fusion clash problem

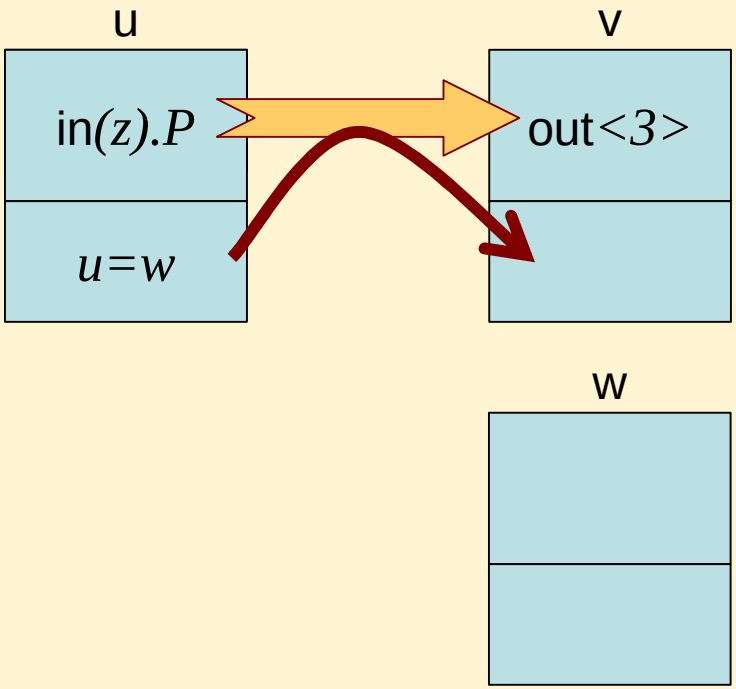


The fusion clash problem:

- If u becomes fused to two different names, how does it know whether to send its code to v or w?

- answer: migrate the fusion down the forwarders until it can be fulfilled; forwarders respect a total order on names.

the fusion clash problem

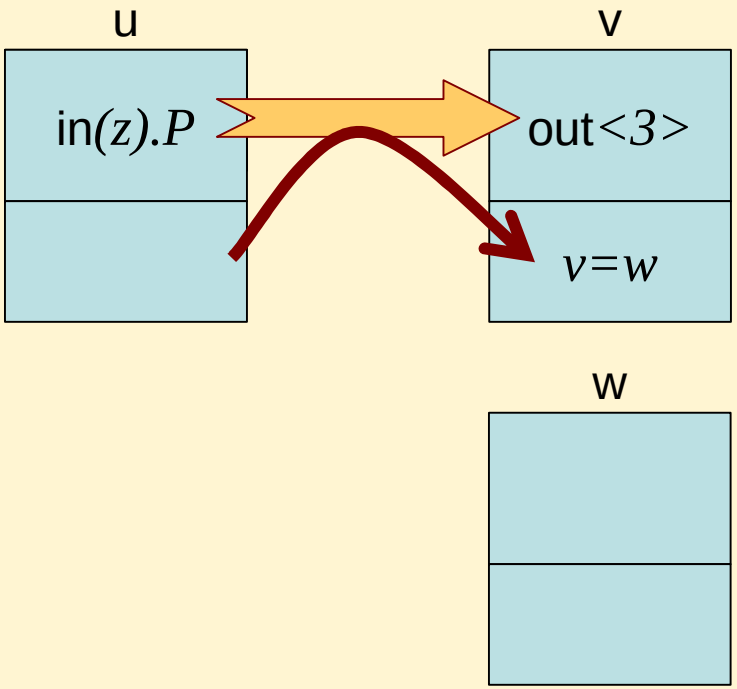


The fusion clash problem:

- If u becomes fused to two different names, how does it know whether to send its code to v or w?

- answer: migrate the fusion down the forwarders until it can be fulfilled; forwarders respect a total order on names.

the fusion clash problem

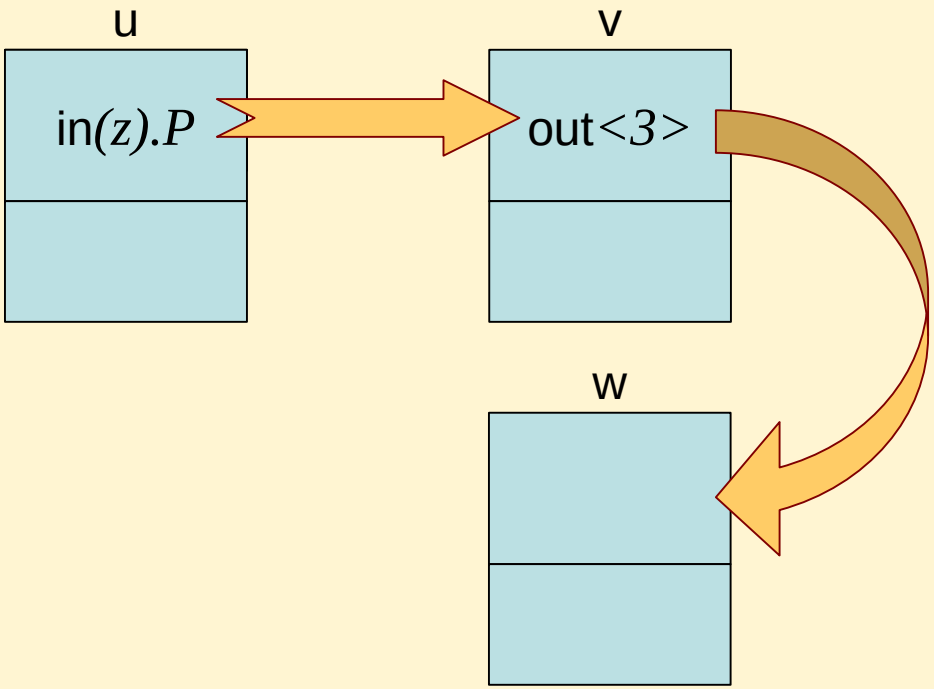


The fusion clash problem:

- If u becomes fused to two different names, how does it know whether to send its code to v or w?

- answer: migrate the fusion down the forwarders until it can be fulfilled; forwarders respect a total order on names.

the fusion clash problem

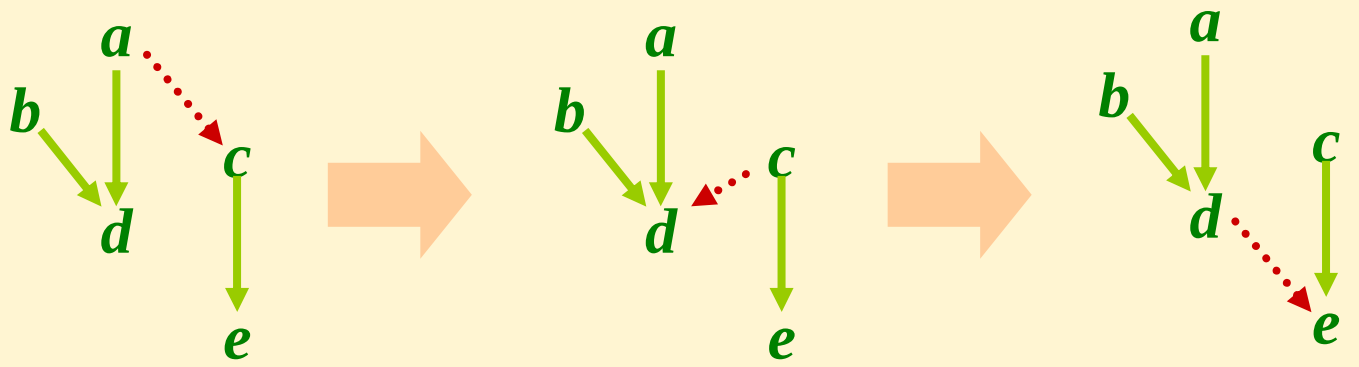


The fusion clash problem:

- If u becomes fused to two different names, how does it know whether to send its code to v or w?

- answer: migrate the fusion down the forwarders until it can be fulfilled; forwarders respect a total order on names.

clash avoidance algorithm

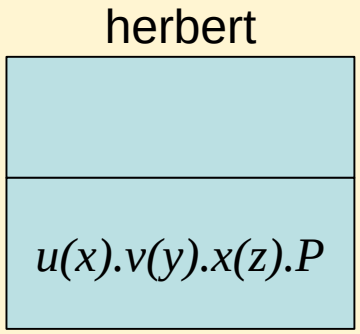


Effect: a distributed, asynchronous algorithm for merging trees.

- Correctness: it preserves the total-order on channels names;
- the equivalence relation on channels is preserved, before and after;
- it terminates, since each step moves closer to the root.

- (similar to Tarjan's Union Find algorithm, 1975)

the input-mobility problem

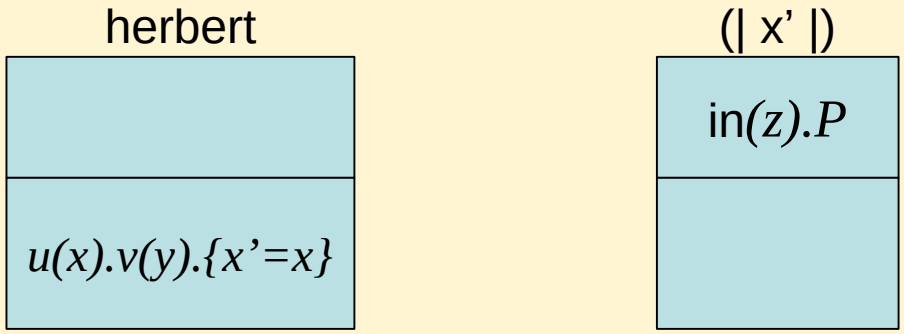


The input-mobility problem:

- how can we pre-deploy $x(z).P$?
  we won't know where it goes until runtime!

- answer 1: well, although $x(z)$ cannot be pre-deployed, at least $v(y)$ and P can be!
- so we can still avoid most of the cost of moving
- Still to do: a mathematical treatment of this.

the input-mobility problem

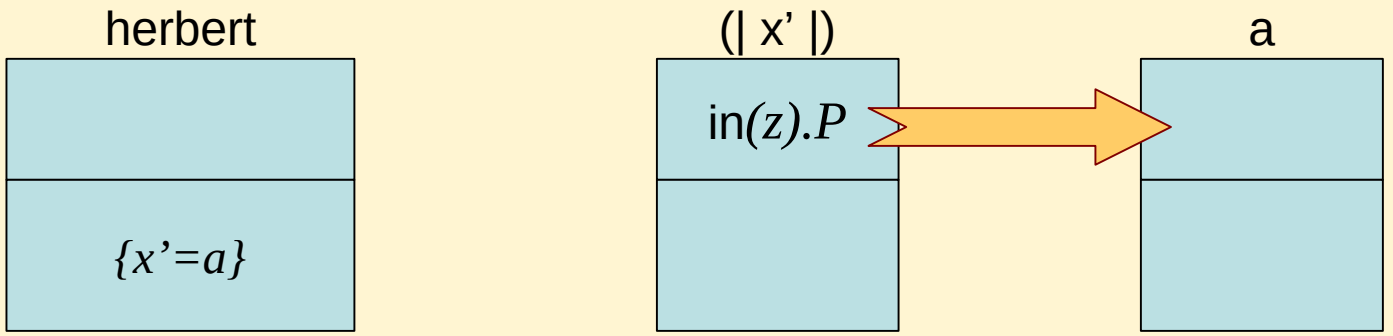


The input-mobility problem: (use fusions!)

- how can we pre-deploy $x(z).P$?
  we won't know where it goes until runtime!

- answer 2: we pre-deploy $x(z).P$ to some private channel x'
- when finally x becomes known to us, we set up a forwarder
- a forwarder allows two channels to be used interchangeably.
- Theorem: efficiency is no worse than Join/Facile.

the input-mobility problem

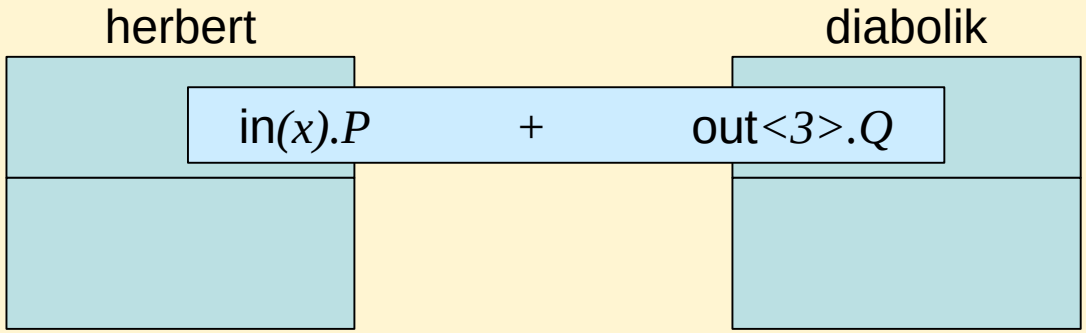


The input-mobility problem: (use fusions!)

- how can we pre-deploy $x(z).P$?
  we won't know where it goes until runtime!

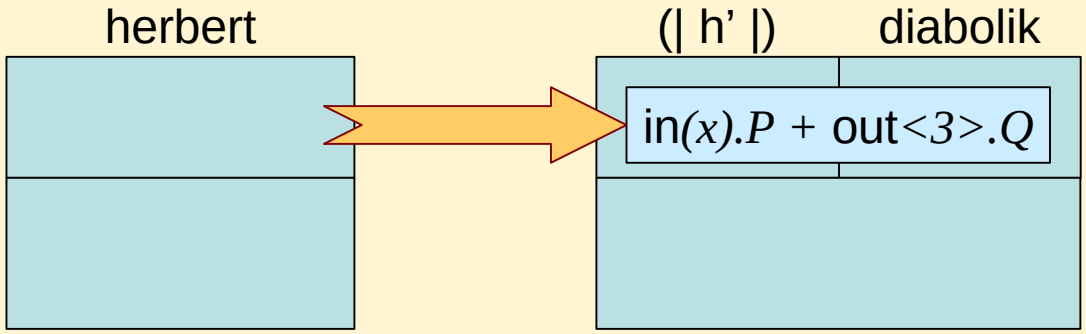
- answer 2: we pre-deploy $x(z).P$ to some private channel x'
- when finally x becomes known to us, we set up a forwarder
- a forwarder allows two channels to be used interchangeably.
- Theorem: efficiency is no worse than Join/Facile.

the choice problem



The choice problem:

- how to implement a distributed choice?
- any proposed reaction at *herbert* would have to ask permission from *diabolik* before proceeding. Awkward.



The choice problem: (use fusions!)

- how to implement a distributed choice?

- suggested answer: a fusion *{herbert=h'}*
  implemented as a forwarder
  so rest of the program can refer equally *herbert* or *h'*
- the local summation is easy to implement.
- But… must fix forwarders if diabolik gets another fusion

The explicit fusion calculus

$$P \quad ::= \quad x{=}y \quad \mid \quad \overline{u}\,\widetilde{x}.P \quad \mid \quad u\,\widetilde{x}.P \quad \mid \quad P|P \quad \mid \quad (x)P \quad \mid \quad \mathbf{0}$$

$$\overline{u}\,\widetilde{x}.P\,|\,u\,\widetilde{y}.Q \quad \longrightarrow \quad \widetilde{x}{=}\widetilde{y}\,|\,P\,|\,Q$$

$x{=}y \mid P \equiv x{=}y \mid P\{y/x\}$	*substitution*
$(x)(x{=}y) \equiv \mathbf{0}$	*local alias*
$x{=}x \equiv \mathbf{0}$	*reflexivity*
$x{=}y \equiv y{=}x$	*symmetry*
$x{=}y \mid y{=}z \equiv x{=}z \mid y{=}z$	*transitivity*

fusion vs pi

reaction in the pi calculus:

$$\overline{u}\,x.P \mid u(y).Q \quad \rightarrow \quad P \mid Q\{x/y\}$$

reaction in the explicit fusion calculus:

$$\begin{array}{lcll}
\overline{u}\,x.P \mid (y)u\,y.Q & \equiv & (y)(\overline{u}\,x.P \mid u\,y.Q) & \textit{assume } y \notin \textit{fn}P \\
& \rightarrow & (y)(x{=}y \mid P \mid Q) & \\
& \equiv & (y)(P \mid x{=}y \mid Q\{x/y\}) & \textit{substitution due to fusion} \\
& \equiv & P \mid (y)(x{=}y) \mid Q\{x/y\} & \textit{scope intrusion} \\
& \equiv & P \mid Q\{x/y\} & \textit{remove local alias}
\end{array}$$

efficiency result

THEOREM

- Using explicit fusions,
  we can compile a program with continuations
  into one without.
- This is a source-code optimisation, prior to execution.
- Every message becomes small (fixed-size).
- This might double the total number of messages but no worse than that. It also reduces latency.
- Our optimisation *is* a bisimulation congruence:

$$C[P] \sim C[\text{optimise}\, P]$$

$$
\begin{array}{ll}
(\text{new}\, xyz, v'@v, w'@w)\,(& \\
\quad ux.\, v'{=}v & \text{// after } u \text{ has reacted, it tells} \\
\quad \mid v'y.\, w'{=}w & \text{// } v' \text{ to fuse to } v\text{, so allowing} \\
\quad \mid w'z & \text{// our } v' \text{ atom to react with } v \text{ atoms} \\
) &
\end{array}
$$

virtual machine, formally

Machines M	::=	$u[B]$	*channel machine at* u
		$(\!(u)\!)[B]$	*private channel machine*
		M, M	
		$\mathbf{0}$	
Bodies B	::=	$\mathsf{out}\widetilde{x}.P$	*output atom*
		$\mathsf{in}(\widetilde{x}).P$	*input atom*
		$!\mathsf{in}(\widetilde{x}).P$	*replicated input*
		P	*pi process*
		$B; B$	

Processes P ::= $\overline{u}\,\widetilde{x}.P \;\mid\; [!]u(\widetilde{x}).P \;\mid\; (x)P \;\mid\; P|P \;\mid\; \mathbf{0}$

a calculus for the implementation

$$u[\mathsf{out}x.P;\mathsf{in}(y).Q] \xrightarrow{\text{react}} u[P;Q\{x/y\}]$$

$$u[\mathsf{out}x.P;!\mathsf{in}(y).Q] \xrightarrow{\text{react}} u[P;Q\{x/y\};!\mathsf{in}(y).Q]$$

$$u[\overline{v}\,x.P]\ v[\,] \xrightarrow{\text{dep.out}} u[\,]\ v[\mathsf{out}x.P]$$

$$u[(\text{new}\,x)P] \xrightarrow{\text{dep.new}} u[P\{x'/x\}]\ (\!|x'|\!)[\,] \quad *$$

$$u[P|Q] \xrightarrow{\text{dep.par}} u[P;Q]$$

$$u[\mathbf{0}] \xrightarrow{\text{dep.nil}} u[\,]$$

* x' fresh, unique

THEOREM $P \sim Q$ iff $u[P] \sim u[Q]$

Are fusions actually useful?

- **yes:** in solving the *continuation problem* as a calculus-friendly way of writing pointers.
- **maybe:** as part of an algorithm for *distributed choice*.
- **maybe:** Cosimo thinks to use *false fusions* like 1=2 as a way to encode failed transactions/compensations.
- **???** Highwire seems to like them. Why?
- **no:** they seem too dangerous and costly, and hard for programmers to grasp intuitively. Seems difficult to mix normal data-types with fusions.

the fusion project at bologna

www.cs.unibo.it/fusion

Lucian Wischik

Cosimo Laneve

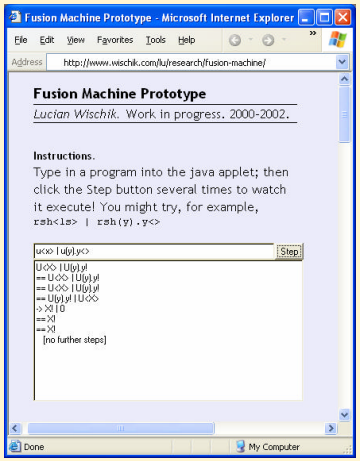
Laura Bocchi

Manuel Mazzara

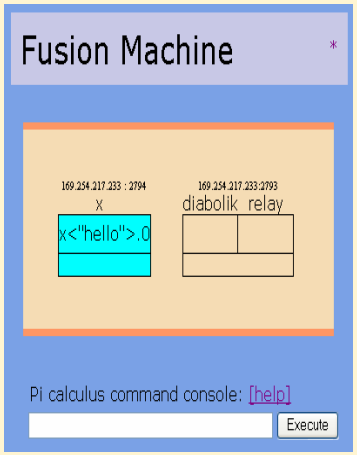
Enrico Tossi

Enrico ?

Lorenzo Agostinelli



prototype implementation in Java



distributed implementation in C++/Win32