

virtual machine, formally

$$\begin{aligned}
 u[\text{out } x.P; \text{in}(y).Q] &\xrightarrow{\text{react}} u[P; Q\{x/y\}] \\
 u[\text{out } x.P; !\text{in}(y).Q] &\xrightarrow{\text{react}} u[P; Q\{x/y\}; !\text{in}(y).Q] \\
 u[\bar{v} x.P] v[] &\xrightarrow{\text{dep.out}} u[] v[\text{out } x.P] \\
 u[(\text{new } x)P] &\xrightarrow{\text{dep.new}} u[P\{x'/x\}] (x')[] \quad * \\
 u[P|Q] &\xrightarrow{\text{dep.par}} u[P; Q] \\
 u[\mathbf{0}] &\xrightarrow{\text{dep.nil}} u[]
 \end{aligned}$$

\*  $x'$  fresh, unique

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

main problem

$$u(x).v(y).w(z).P \mid \bar{u}a \mid \bar{v}b \mid \bar{w}c$$

**Q.** Example will transport all of  $P$  first to  $u$ , then  $v$ , then  $w$ .  
How to implement this more efficiently?

**?A.** Guard  $P$  and then, at the last minute, transport  $P$  direct to its final destination. (Parrow, 1999). But this causes a latency problem...

$$(\text{new } t)(u(x).v(y).w(z).\bar{t}xyz \mid t(xyz).P)$$

### main problem

$$u(x).v(y).w(z).P \mid \bar{u}a \mid \bar{v}b \mid \bar{w}c$$

**Q.** Example will transport all of  $P$  first to  $u$ , then  $v$ , then  $w$ .  
How to implement this more efficiently?

**A.** Optimistically send  $P$  to its expected final destination. Use **explicit fusions** (Gardner and Wischik, 2000). Then, if we had sent it to the wrong place, it will become **fused** to the correct place and it can **migrate**...

### the explicit fusion calculus

$$P ::= x=y \mid \bar{u}\tilde{x}.P \mid u\tilde{x}.P \mid P|P \mid (x)P \mid \mathbf{0}$$

$$\bar{u}\tilde{x}.P \mid u\tilde{y}.Q \longrightarrow \tilde{x}=\tilde{y} \mid P|Q$$

$$x=y \mid P \equiv x=y \mid P\{y/x\} \quad \text{substitution}$$

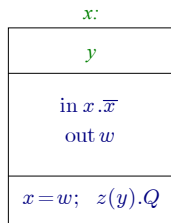
$$(x)(x=y) \equiv \mathbf{0} \quad \text{local alias}$$

$$x=x \equiv \mathbf{0} \quad \text{reflexivity}$$

$$x=y \equiv y=x \quad \text{symmetry}$$

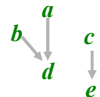
$$x=y \mid y=z \equiv x=z \mid y=z \quad \text{transitivity}$$

### fusion machine



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

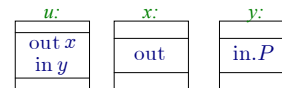
Collectively, the fusion pointers make a *forest* which respects a total order on names:



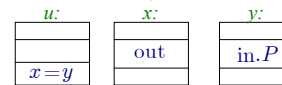
### the explicit fusion calculus

$$\bar{u}x \mid uy \mid \bar{x} \mid y.P \longrightarrow x=y \mid \bar{x} \mid y.P$$

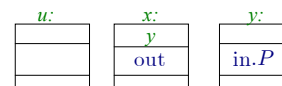
$$\longrightarrow x=y \mid P$$



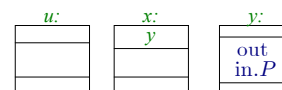
↓ **react**



↓ **deploy fusion by sending to  $x$  the message "fuse yourself to  $y$ "**



↓ **migrate atom from  $x$  to  $y$**



### THEOREM

$$P \sim Q \text{ iff } u[P] \sim u[Q]$$

## fusion results

### 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:  
 $C[P] \sim C[\text{optimise } P]$

```
(new xyz, v'@v, w'@w) (
  ux. v'=v      // after u has reacted, it tells
| v'y. w'=w    // v' to fuse to v, so allowing
| w'z          // our v' atom to react with v atoms
)
```

## what we are discovering

### THOUGHTS

- Channel-based makes for *easy implementation*. (I have implemented it in java and C++. Students have implemented it in Jocaml and Prolog). Also makes for *easy and strong proofs of correctness*.
- Fusions allow for *optimisation* at source level, by “pre-deploying” fragments to their expected destination.
- The machine is just a start. Substantial work needed to build a *full implementation* and *language* on top of it... XML data types (Mazzara, Meredith). Transactions and rollbacks like Xlang. This is motivated by the problem of ‘false fusions’ like  $2=3$ , and seems the best way to deal with failure (Laneve, Wischik, Meredith). Quantify the cost of fusion/migration.

## Supplemental Slides

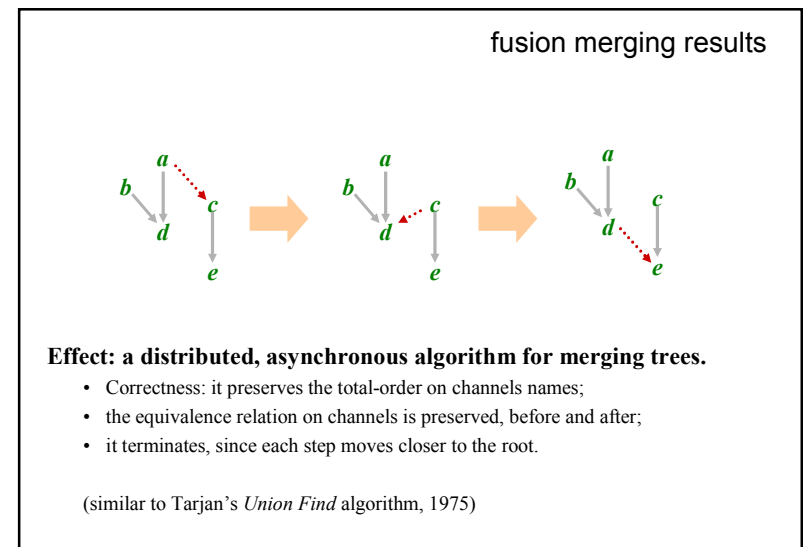
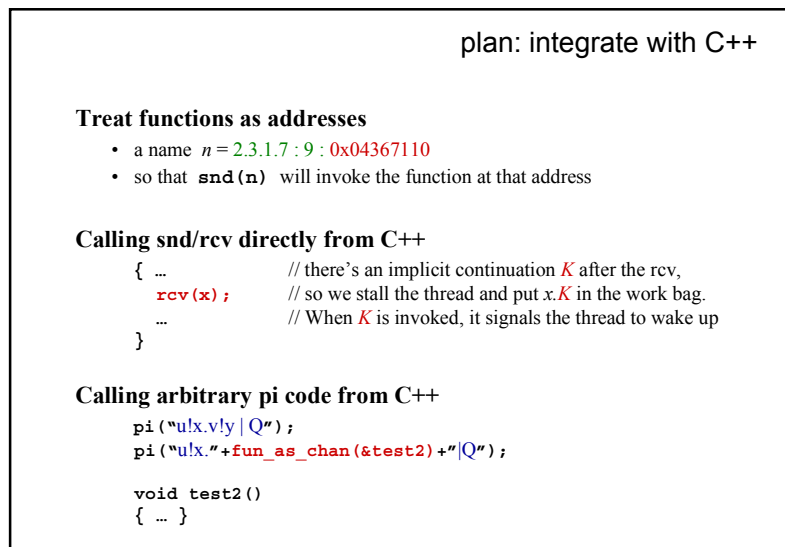
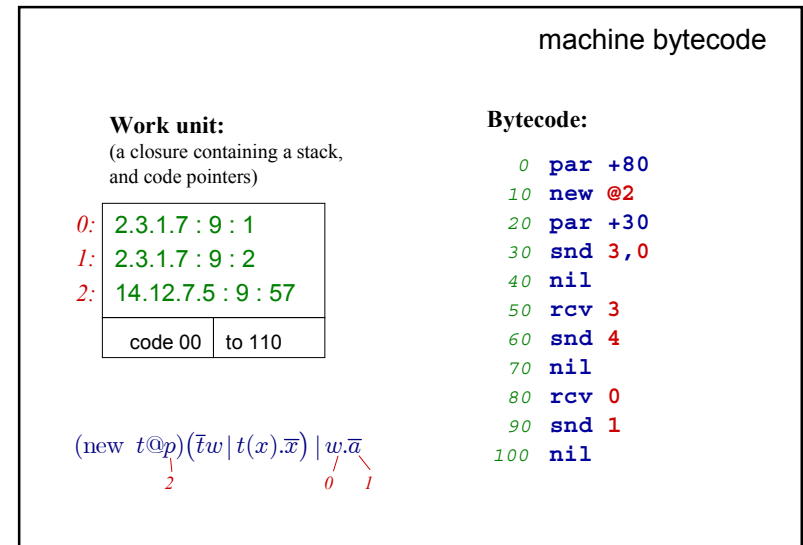
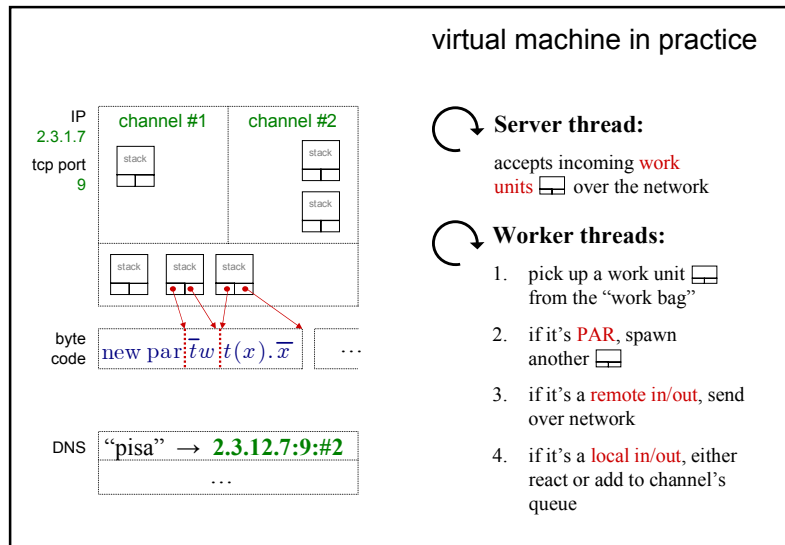
Grammar for fusion machine calculus  
 Implementation notes  
 Fusion algorithm

## virtual machine, formally

Machines  $M ::= u[B] \quad \text{channel machine at } u$   
 $(u)[B] \quad \text{private channel machine}$   
 $M, M$   
 $0$

Bodies  $B ::= \text{out}\tilde{x}.P \quad \text{output atom}$   
 $\text{in}(\tilde{x}).P \quad \text{input atom}$   
 $!\text{in}(\tilde{x}).P \quad \text{replicated input}$   
 $P \quad \text{pi process}$   
 $B; B$

Processes  $P ::= \bar{u}\tilde{x}.P \mid [!u(\tilde{x}).P \mid (x)P \mid P|P \mid 0$



## fusion merging algorithm

$u[x=y] x[p:] \xrightarrow{\text{dep.fu}} u[] x[y: y=p]$  \* assuming  $x < y$   
\* if  $p$  was nil, then  
discard  $y=p$  in the result

The explicit fusion  $x=y$  is an *obligation to set up a fusion pointer*.

A channel will either fulfil this obligation (if  $p$  was nil), or will pass it on.