```
to Point x y | other
    ox  →  (← →(^:x)  ^x)
    oy  →  (← →(^:y)  ^y)
    -   →  (:other. ^(Point (x - other x) (y - other y)))
    +   →  ...

to Number
    @   →  (:n. ^(Point SELF n))
```

Behaviours (in SLS) have ISTR 3 slots — or similar

- superclass
- slot count
- method table → sorted somehow. array with evens = keys, odds = values

Point ~~class~~ is a Class

Behaviour slots.

SUPER: Object	0
SLOT COUNT: 2	1
METHODS	2
SLOT NAMES: {'x'. 'y'}	3
NAME: 'Point'	4
ORGANISATION: ...	5
etc.	...

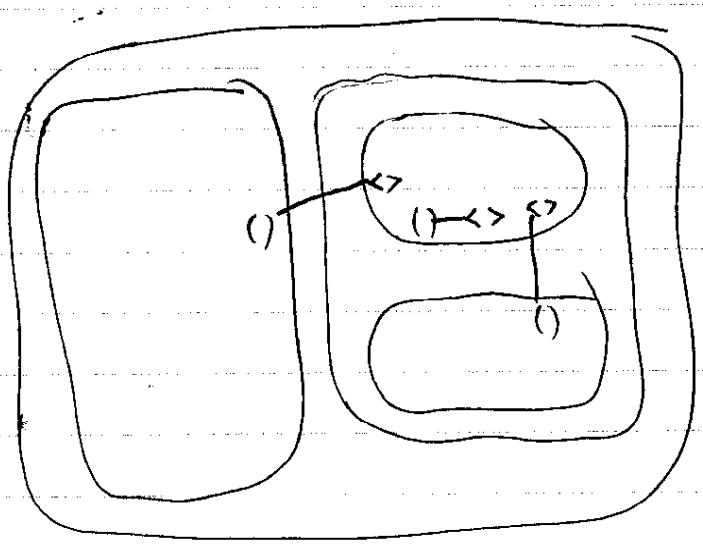
0	x	0
2	x:	1
4	y	2
6	y:	3
8	-	4
10	+	5
12.		...

but sorted somehow for binary searching

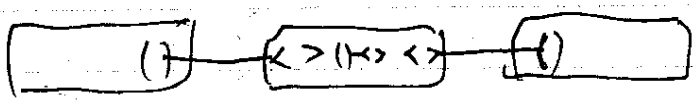
```
primitiveNew = λ behaviour, n.  x ← alloc headersize + b[1] + n
                                x.class ← b
                                [A: x?k,msg,args. [DISPATCH!k,msg,A] fork.
                                x

A ← [x?k,msg,args; A fork; DISPATCH!k,msg,args]; A fork.
```

LINDA: IN/OUT/EVAL



0
$(x)P$
$P \mid Q$
$\mu A.P$
A
$x\langle M \rangle$
$x(M).P$
$x[P].Q$
lift $x, y(z).P$
drop x, y



Canonical form – news all on the outside

Problem with sends: not just giving them a location, but exposing that location to reflection – TUPLE SPACE w. the space the unit of reflection

No locationless code??

Dist. join – how is migration handled?

$\nu \tilde{x}.\, x[P] \mid \ldots$

Let $x[M] \equiv x\langle M \rangle$? Say M were names only. → names floating free in x. $x(M)$ then interpreted as pulling a value from x. (a value in this context)

$\langle x \Leftarrow M \rangle$ as in π^* is an input not an output.