Our adaptation of Linda:

```
in( x, y, ?z )

in( "AT", ?b, 7 )
```

- "AT" ... 7 — match on these
- ?b — assign to b

Not compatible with Python!

methods can't assign to variables in arguments

New Syntax

```
In(( "AT", None, 7 ))
```

- extra parentheses to make tuple argument
- "AT" ... 7 — match on these
- None — no match
- In — block until match, then remove tuple and return tuple

```
Inp(( "AT", None, 7 ))
```

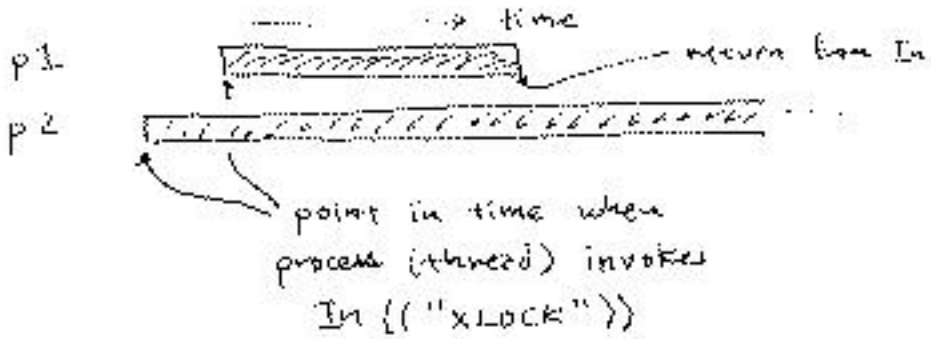
- Inp — non blocking, but raises an exception if tuple not found.

Examples of Locks, Semaphores in Linda

First, let's consider how In and Out work with concurrency

Suppose ("XLOCK") is the only tuple in tuplespace, and two processes (threads) try In(("XLOCK")) concurrently

-- Both match the only tuple
-- Both want to remove the tuple

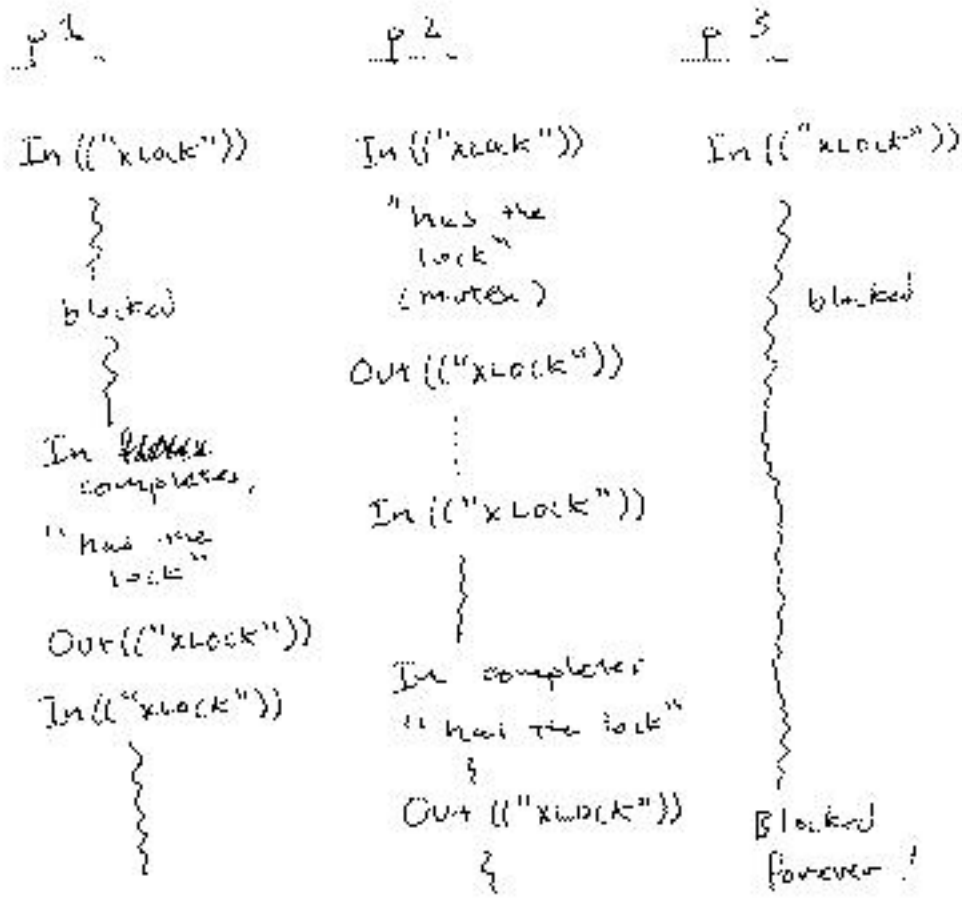


Notice:

1. No "FIFO" guarantee
2. Only one process gets the tuple -- other processes just blocks

Lock implementation:

Initially, there is exactly one ("xlock") in tuple space, then:

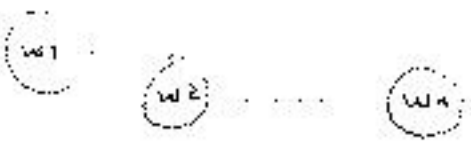


Notice NO "FAIRNESS" guarantee.

Semaphore is easy generalization of this.

Master – many worker Paradigm

puts tuples into the space that represent commands or tasks for workers



n workers, each competes to find a task in tuple space, & perform it.

Master:

```
for i in range(m):
    Linda.Out(("task", description(i)))
```

Worker

```
v = Linda.In(("task", None))
perform v[1]
```

loop

This is like a "parallel" for loop.