RCHAIN COOPERATIVE

At all scales

Lucius Gregory Meredith

Perm Winter School 2018

RCHAIN
COOPERATIVE

What is scale?

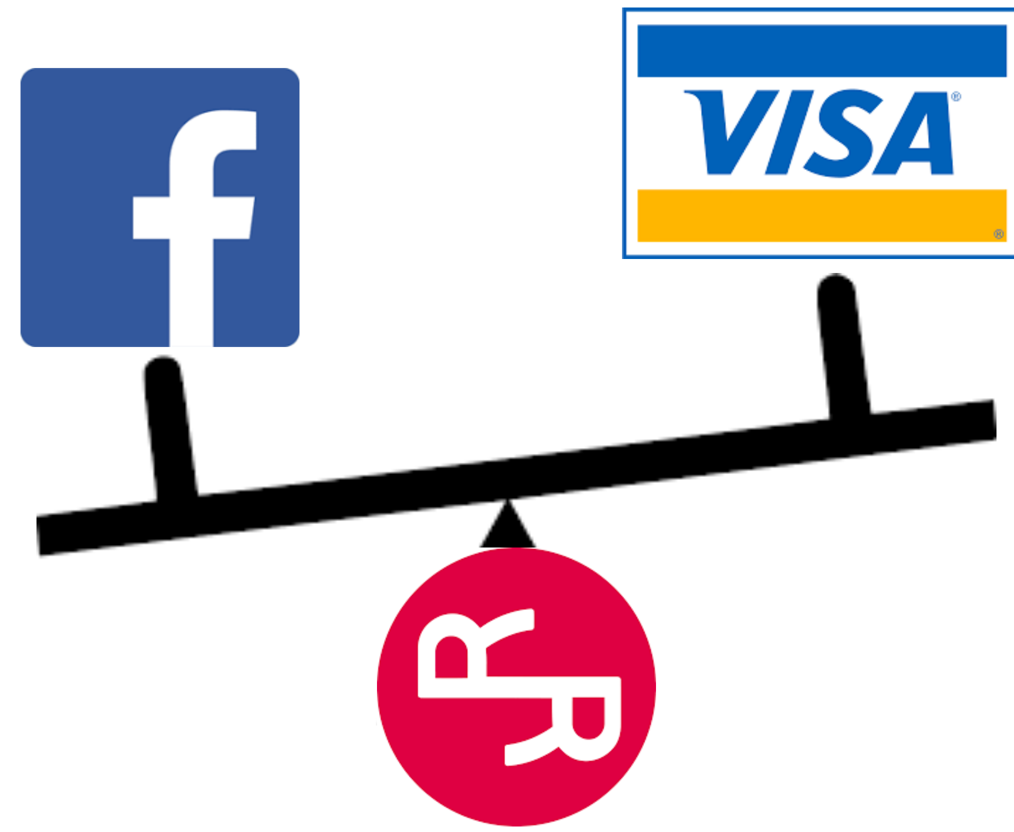
Is scale throughput?

Is scale correctness?

Is scale adoption?

Is scale throughput?

If we are building an economically secured sensorship-resistant, public blockchain for the purpose of global coordination, then scale includes throughput.



In fact, we can articulate the throughput requirements. Such a blockchain must provide throughput ceteris paribus with the existing coordination infrastructure. It must be on par with Visa and with Facebook.

Is scale correctness?

If we are building an economically secured sensorship-resistant, public blockchain for the purpose of global coordination, then scale includes correctness.

Firstly, it doesn't matter how many transactions / sec the infrastructure conducts if they are not correct. Producing garbage faster just creates more garbage.

More importantly, the public has to trust the chain. If they are going to use it to coordinate, they have to rely on it.

Is scale adoption?

If we are building an economically secured sensorship-resistant, public blockchain for the purpose of global coordination, then scale includes adoption.

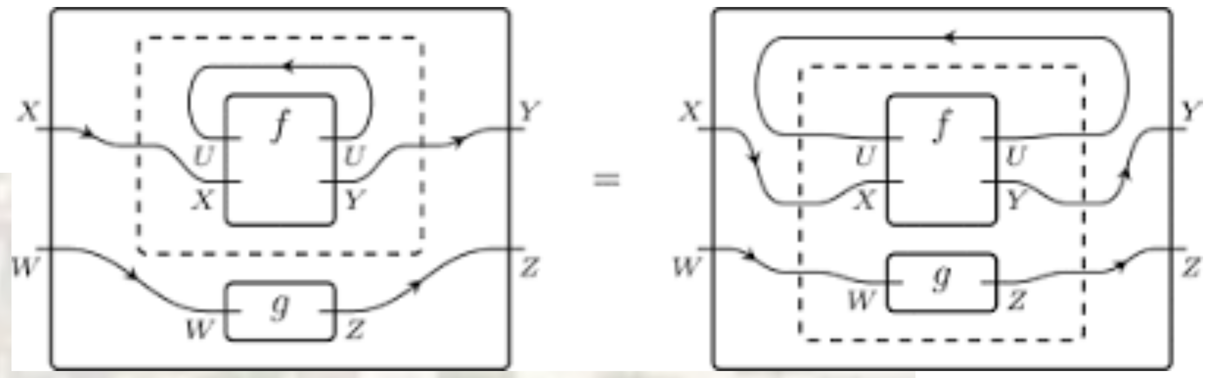
If we build it, will they come?

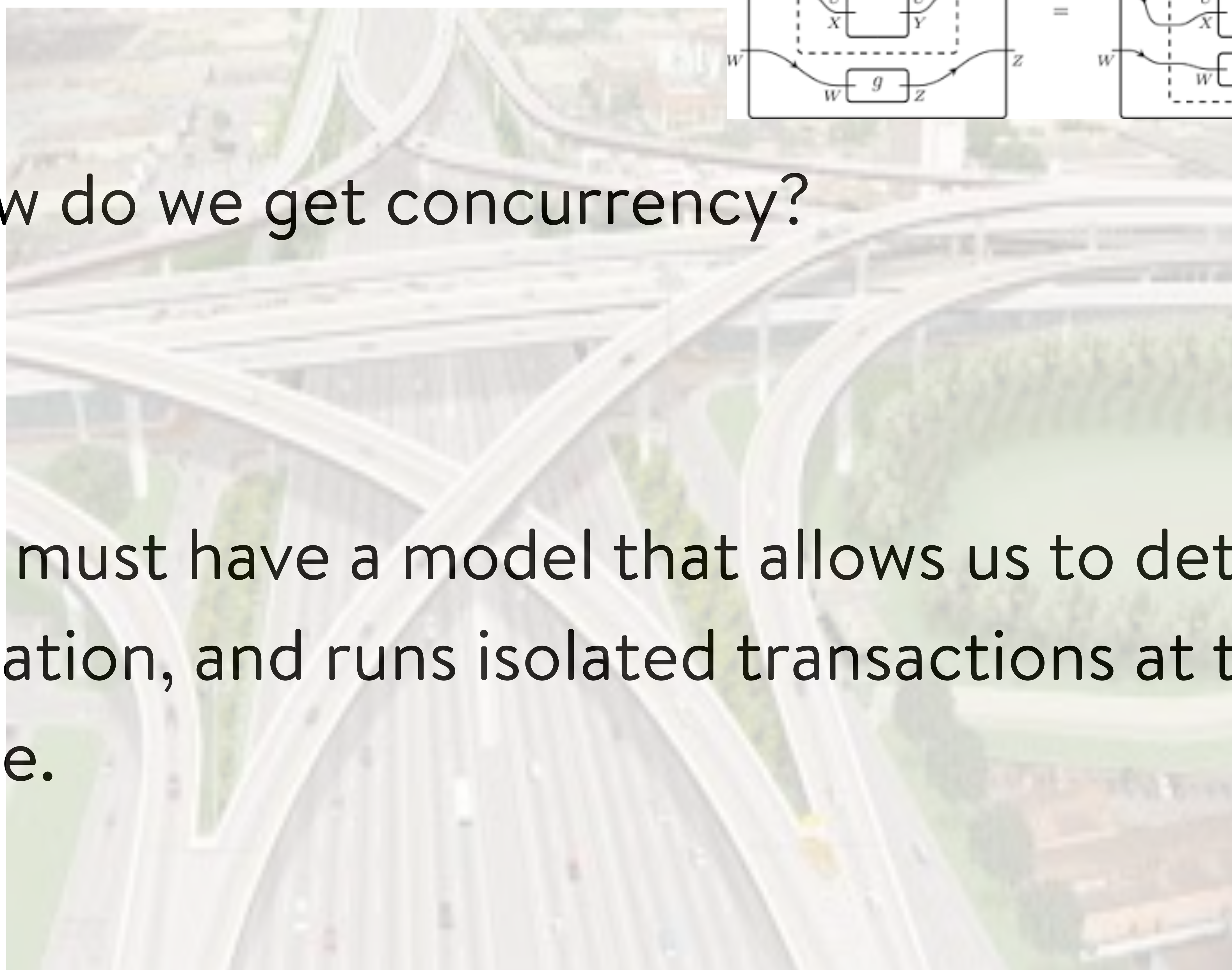
In reality, the technology is going to be commoditized. Soon, it will be as easy as pressing a button to standup a scalable blockchain. Community development, adoption, and network effects will be more significant differentiators.

That's all there is to it, really.

How do we get throughput?

Concurrency!





How do we get concurrency?

We must have a model that allows us to detect isolation, and runs isolated transactions at the same time.

How do we get correctness?

Not very open source

By construction!

Wait, what is correct-by-construction?

A lot of platforms are like this

One way is begin with a formal model of computation that we know to be consistent with requirements and then prove correct each step of refinement toward implementation.

Alternatively, we simply let the formal model be the implementation!

The rho-calculus begets both the RhoVM and rholang

How do we get correctness?

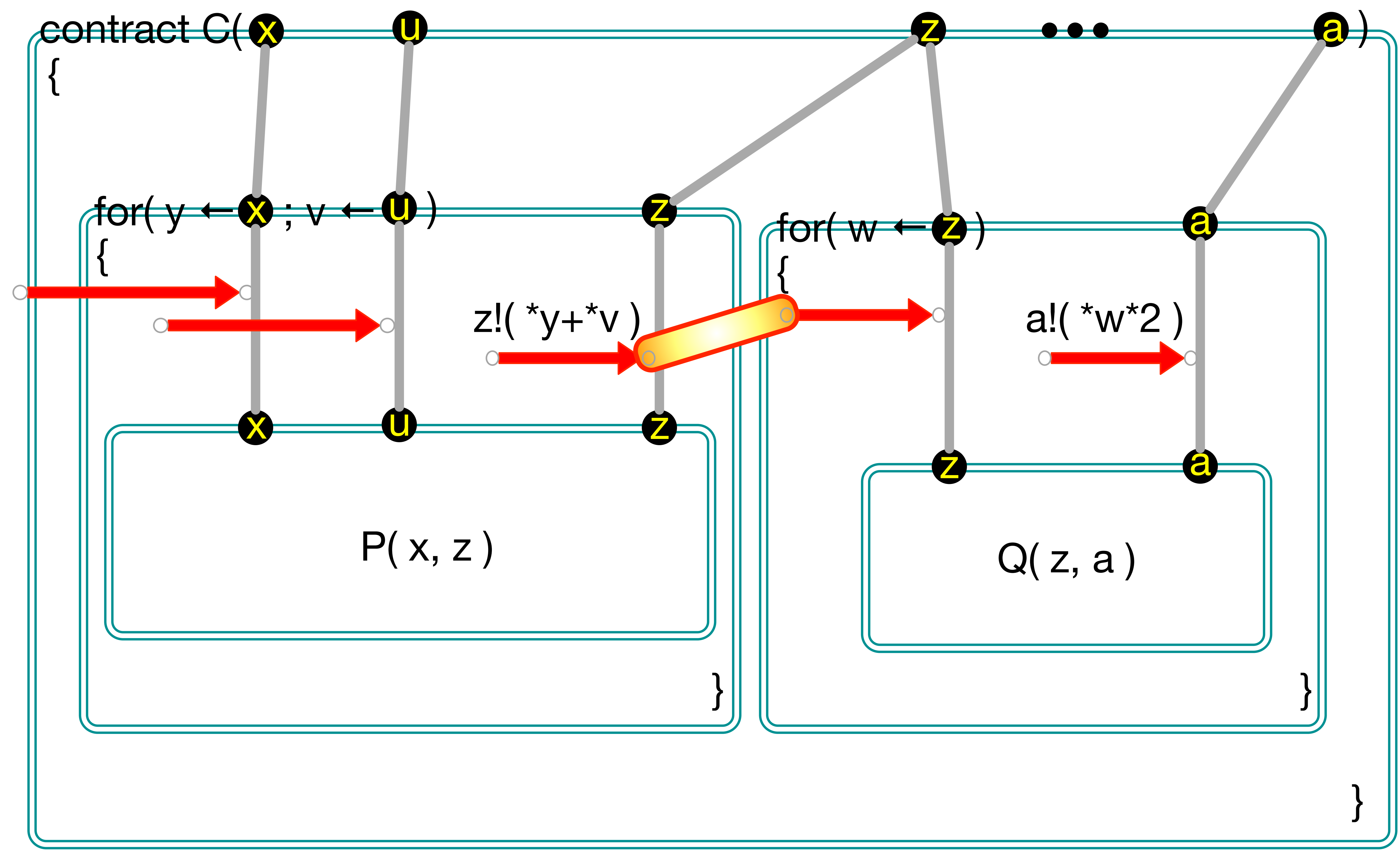
We use types to ensure the relationship between requirements and implementation.

The LADL algorithm takes a language spec and a collection type and generates a sound type system.

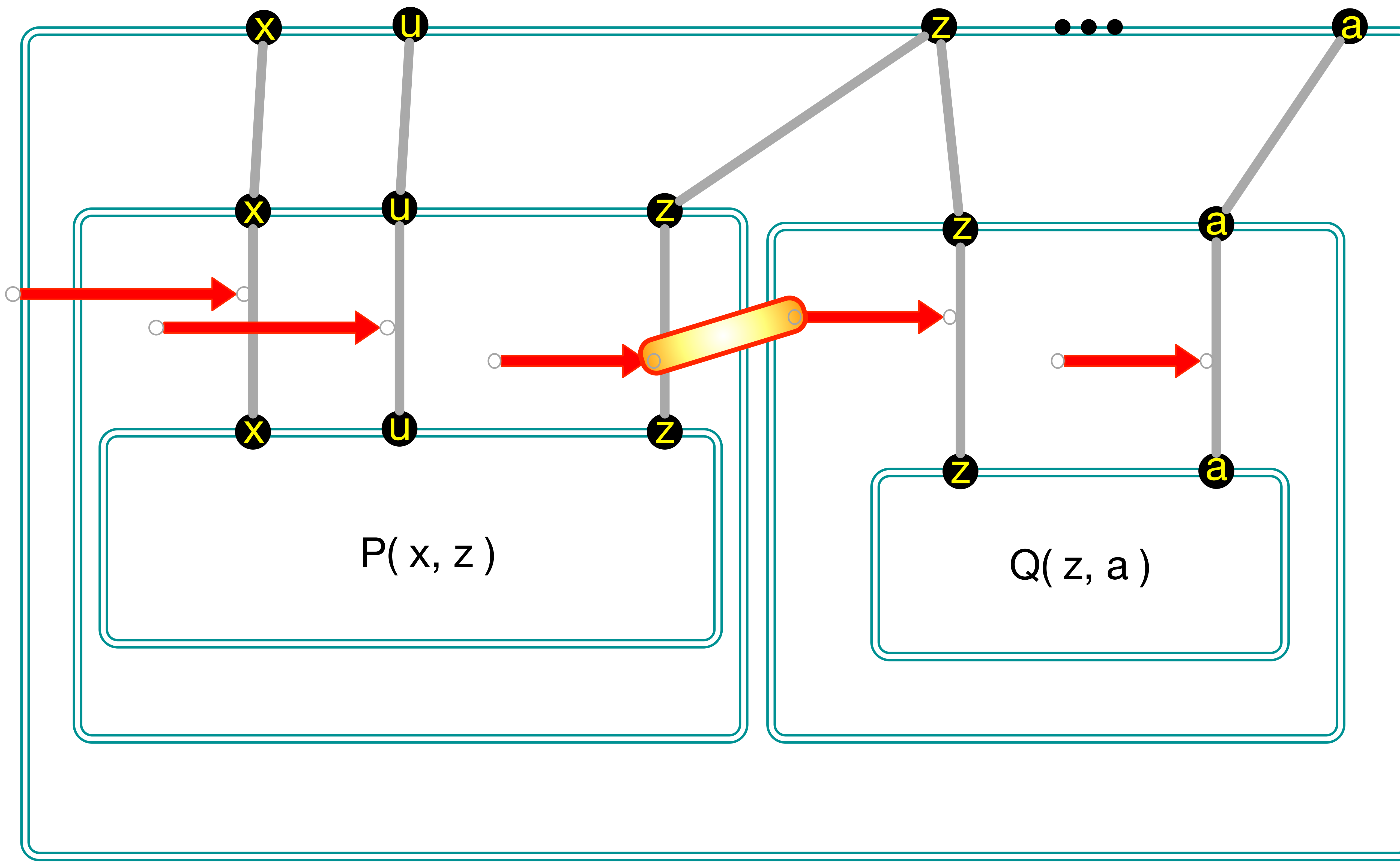
Rholang’s types are generated by the LADL algorithm.

A simple contract: hello arithmetic

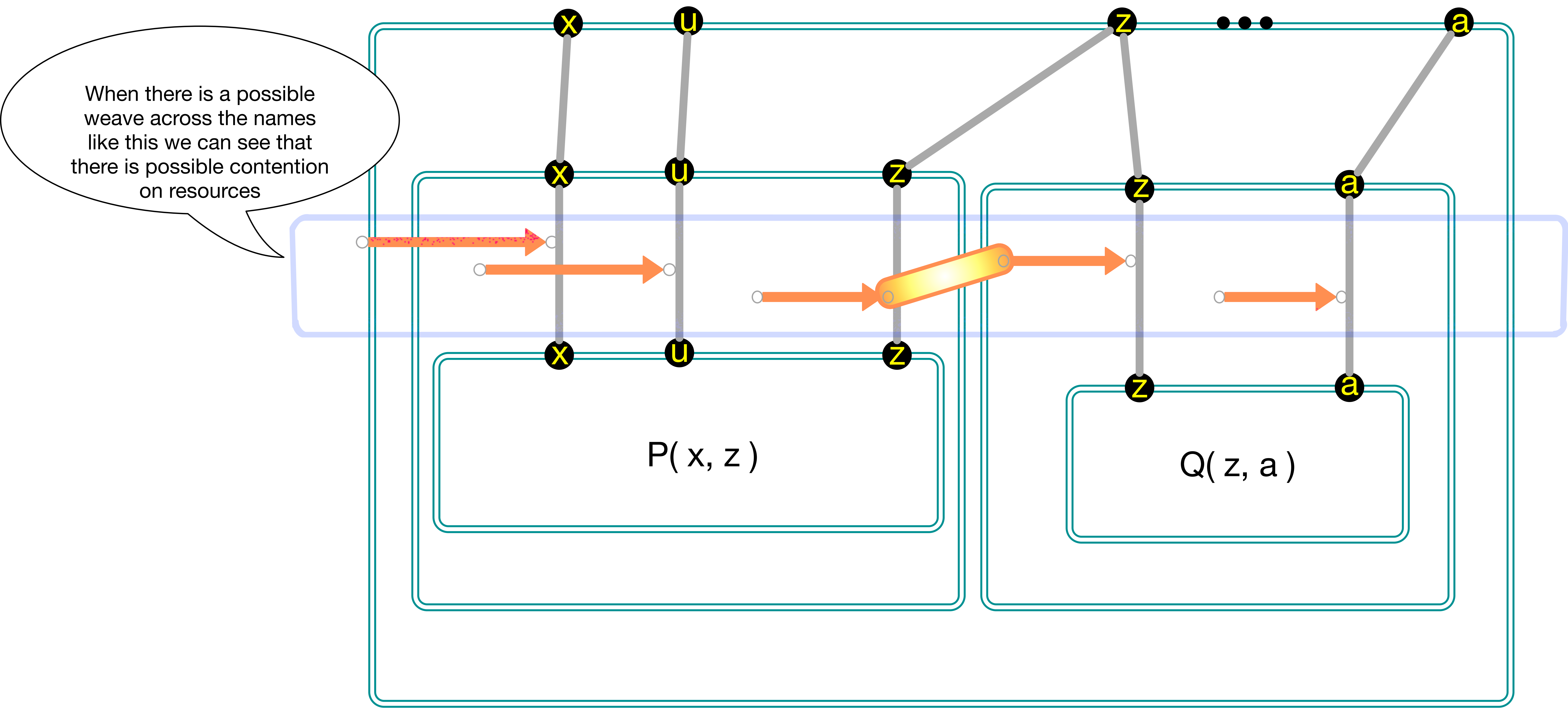
```
contract C( x, u, z, a ) = {
   for( y ← x; v ← u ){ z!( *y + *v ) | P( x, z ) }
   | for( w ← z ){  a!( *w*2 ) | Q( z, w ) }
}
```



for(y ← x; v ← u){ z!(*y+*v) | P(x, z) } | for(w ← z){ a!(*w*2) | Q(z, w) }

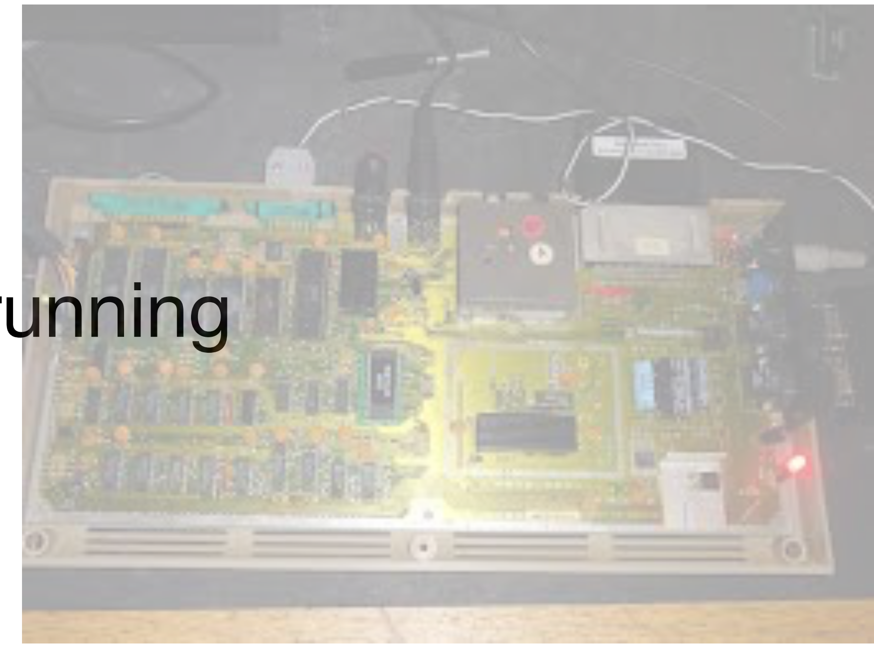


Every contract is actually a generalized braid or tangle



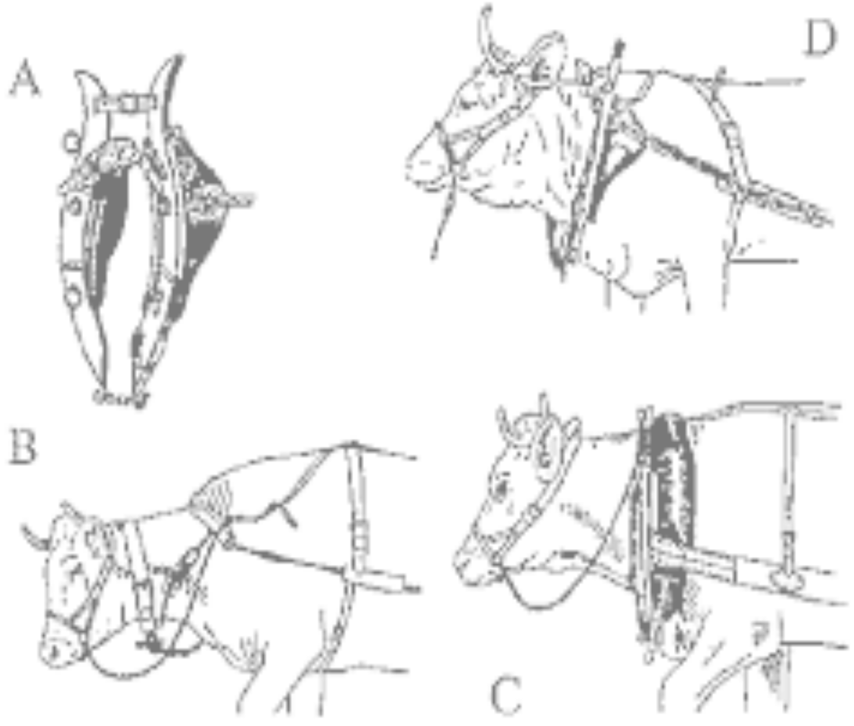
Every contract is actually a generalized braid or tangle

A test harness

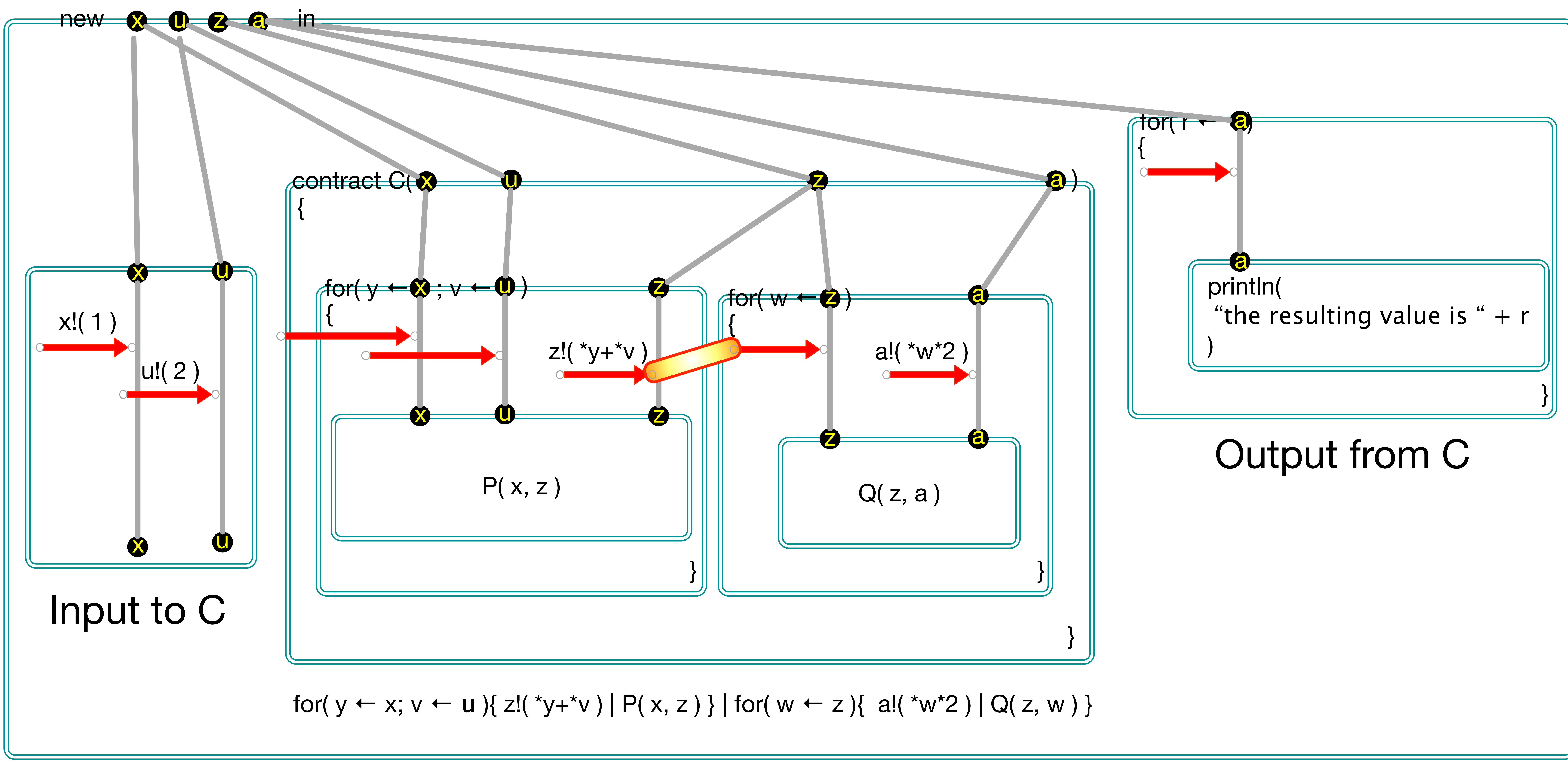


Setting P(x, z) = 0 and Q(z, w) = 0 and running

new x, u, z, w, a in
  x!(1) | u!(2) | C(x, u, z, w, a)
  | for(r ← a){ println(“the resulting value is “ + r) }



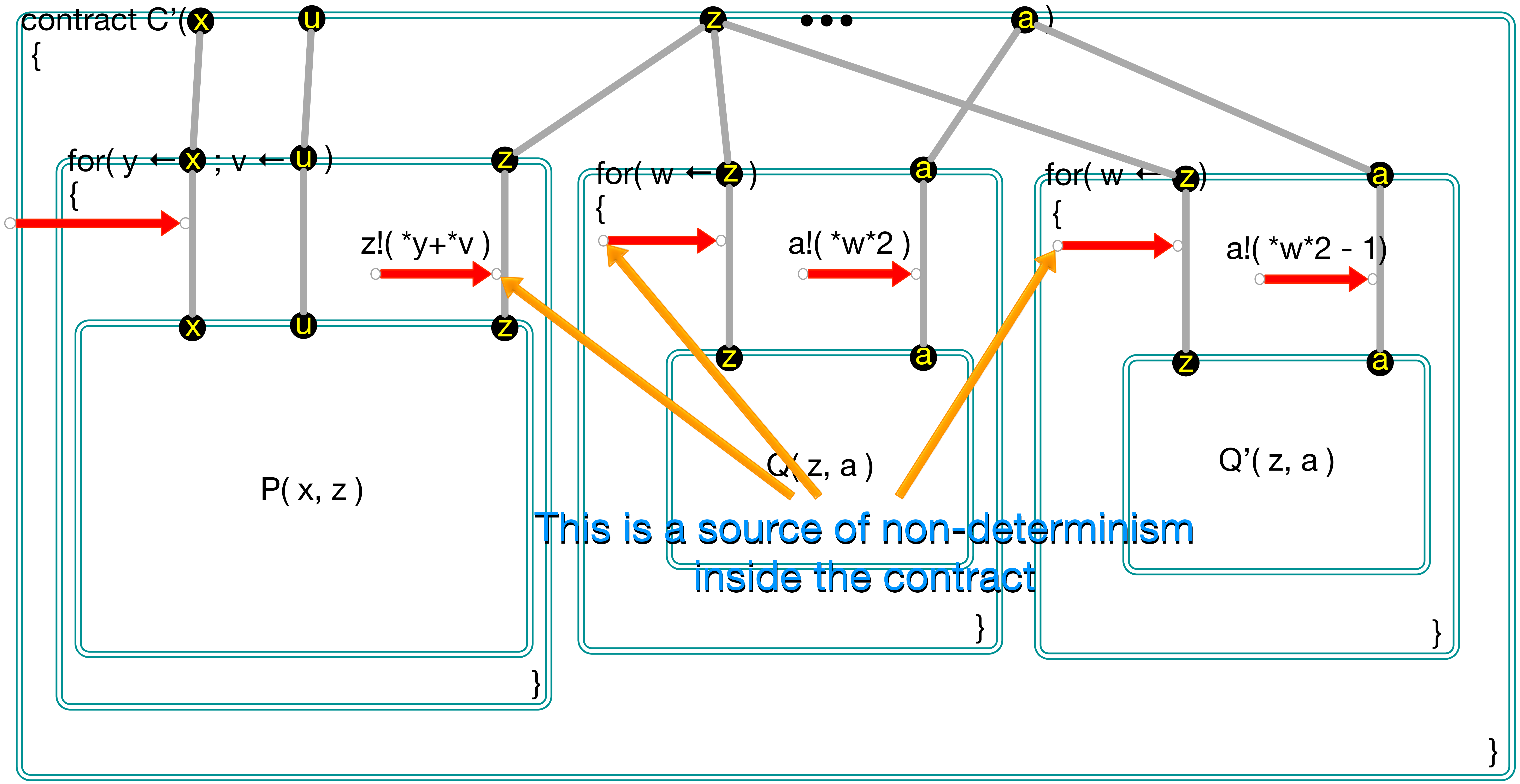
Will print “the resulting value is 6”



for(y ← x; v ← u){ z!(*y+*v) | P(x, z) } | for(w ← z){ a!(*w*2) | Q(z, w) }

A non-deterministic contract

```
contract C'( x, u, z, a ) = {
   for( y ← x; v ← u ){ z!( *y + *v ) | P( x, z ) }
   | for( w ← z ){  a!( *w*2 ) | Q( z, w ) }
   | for( w ← z ){  a!( *w*2 - 1 ) | Q( z, w ) }
}
```



}

for(y ← x){ z!(*y+1) | P(x, z) }
| for(w ← z){ a!(*w*2) | Q(z, w) }
| for(w ← z){ a!(*w*2 - 1) | Q'(z, w) }

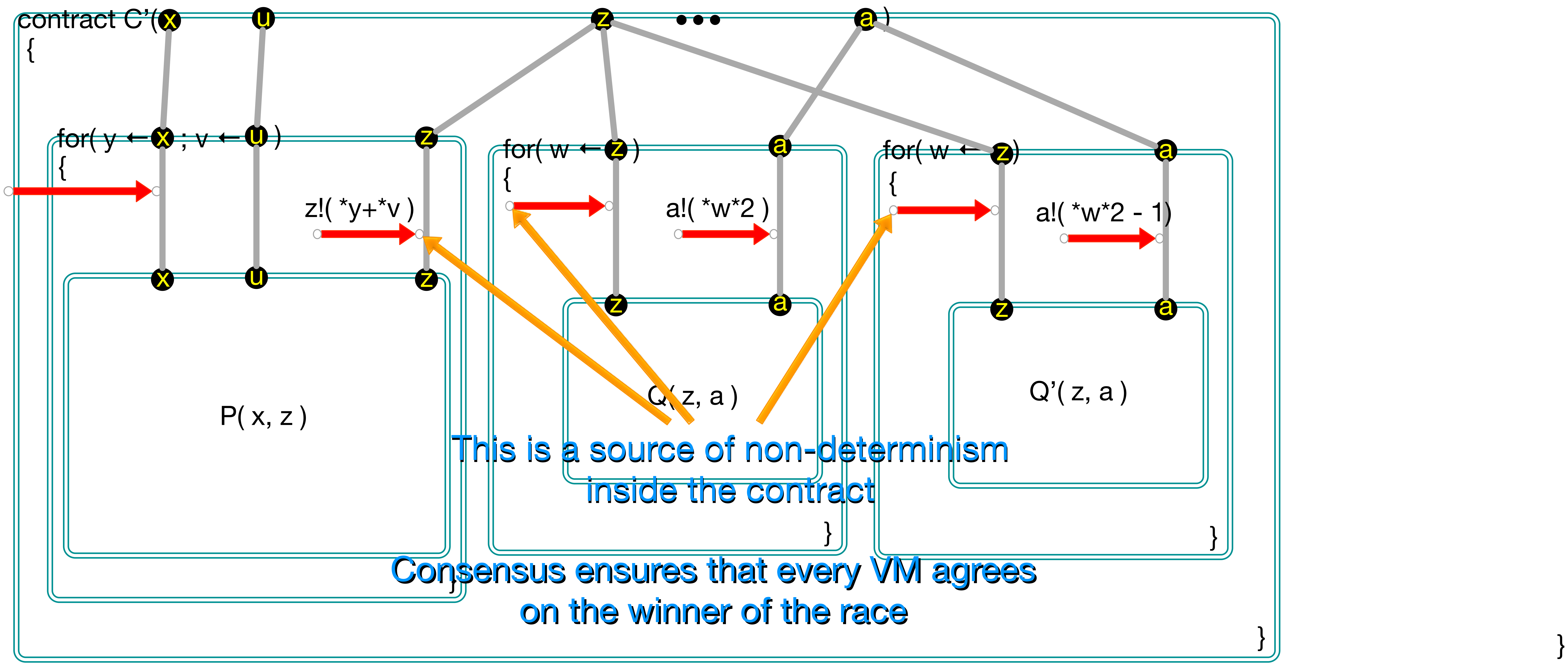
Generalizing the test harness

Setting P(x, z) = 0 and Q(z, w) = 0 and running

K = new x, u, z, w, a in
x!(1) | u!(2) | []
| for(r ← a){ println(“the resulting value is “ + r) }

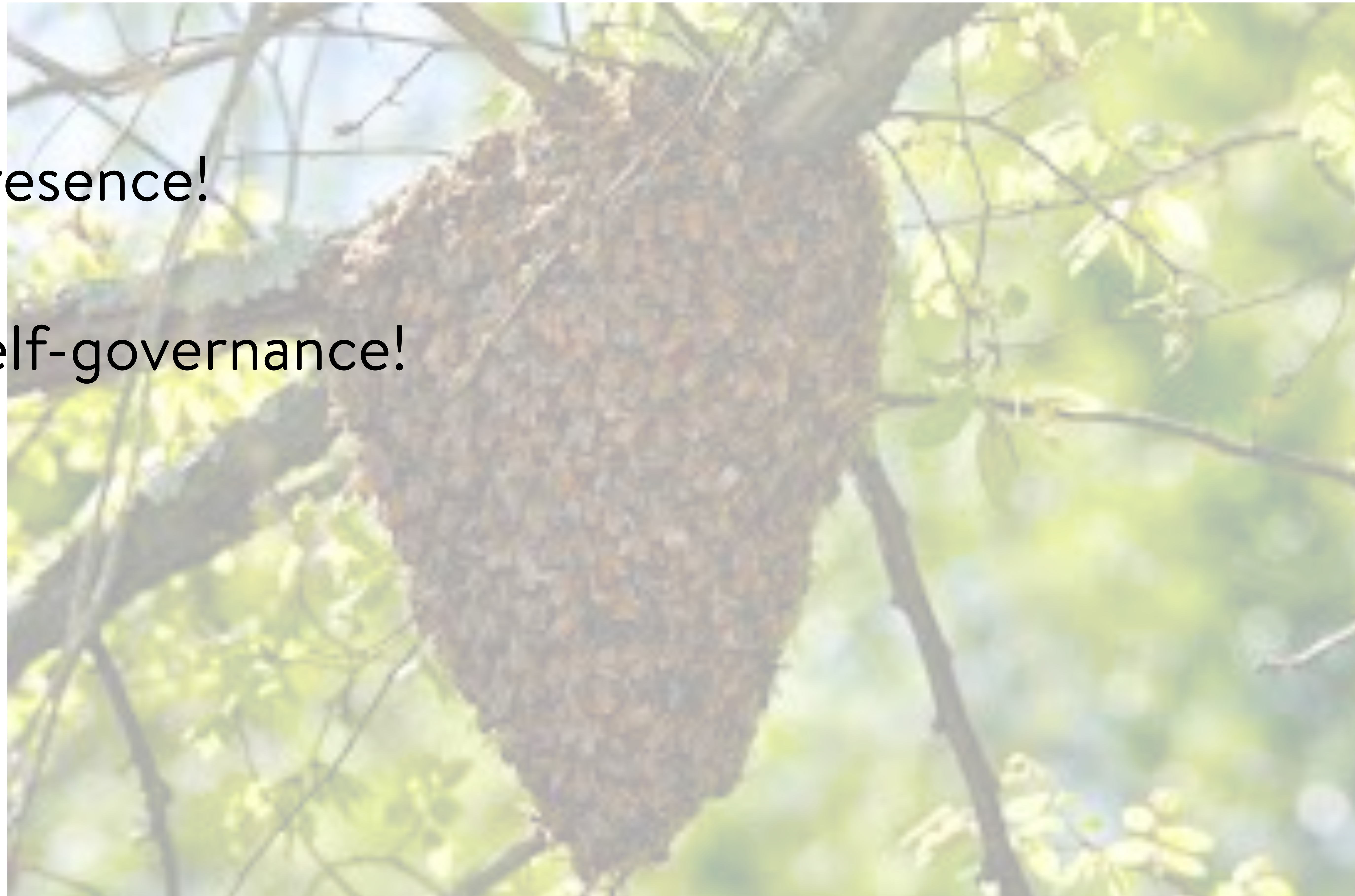
K[C(x,u.z.w.a)] Will print “the resulting value is 6”

K[C’(x,u.z.w.a)] Will either print “the resulting value is 6”
or “the resulting value is 5”
non-deterministically



for(y ← x){ z!(*y+1) | P(x, z) }
| for(w ← z){ a!(*w*2) | Q(z, w) }
| for(w ← z){ a!(*w*2 - 1) | Q'(z, w) }

How do we get adoption?



Presence!

Self-governance!

$$
\begin{array}{lll}
P,Q ::= & 0 & \text{stopped process} \\
\mid & \texttt{for}(\ y_1 \leftarrow x_1;\ \ldots;\ y_N \leftarrow x_N\ )P & \text{input-guarded continuation} \\
\mid & x!(\ Q\ ) & \text{output} \\
\mid & P \mid Q & \text{parallel composition} \\
\mid & {*}x & \text{deref} \\
\mid & (\texttt{new}\ x)P & \text{fresh channel} \\
\mid & (\texttt{new}\ G)P & \text{fresh group}
\end{array}
$$

$$x,y ::= @P$$

$$\texttt{for}(\ y_1 \leftarrow x_1;\ \ldots;\ y_N \leftarrow x_N\ )P \mid x_1!(\ Q_1\ ) \mid \ldots \mid x_N!(\ Q_N\ ) \rightarrow P\{@Q_1/y_1,\ldots,@Q_N/y_N\}$$

$T,U ::= B_1 \mid \ldots \mid B_M \mid G[T_1,\ldots,T_K]\backslash H$ channel type

$H ::= \varnothing \mid G::H$ effect

$P,Q ::= \ldots$

$\mid \; for(\, y_1 : T_1 \leftarrow x_1; \ldots; y_N : T_1 \leftarrow x_N \,)P$ input-guarded continuation

$\mid \; \ldots$

$E \vdash P:H$ typed process

```
P,Q ::= 0                     stopped process
     |  U( x )                location update
     |  for( y <- x )P        input-guarded continuation
     |  x!( Q )               output
     |  P|Q                   parallel composition
     |  *x                    deref
     |  COMM( K )             situation catalyst

x,y ::= @<K,Q>
K   ::= [] | for( x <- y )K | x!( K ) | P|K

COMM( K ) | for( x <- y )P | x!( Q ) -> P{@<K,Q>/y}
U( x ) | *@<K,Q> -> COMM( K ) | x!( Q )

*y{@<K,Q>/y} = K[Q]
```

RCHAIN COOPERATIVE

A simple calculation shows….

```
for(@<K,Q> <- @<[],0>)*@<K,Q> | @<[],0>!(P) | COMM(K')
-> *@<K,Q>{@<K',P>/@<K,Q>} = K'[P]
```

If K' is of the form

```
for(@<K,Q> <- @<[],0>)*@<K,Q> | @<[],0>!([]) | COMM(K') | …
```

then P moves through contexts