Fishing for a good runtime starts with fusion

Amos Robinson, UNSW

(Image: Jason Rogers' flickr)

Fusion

```
filterMax (xs : [Int])
 = let   above= filter   (>0)   xs
         m      = max            xs
   in    (above, m)
```

Clustering

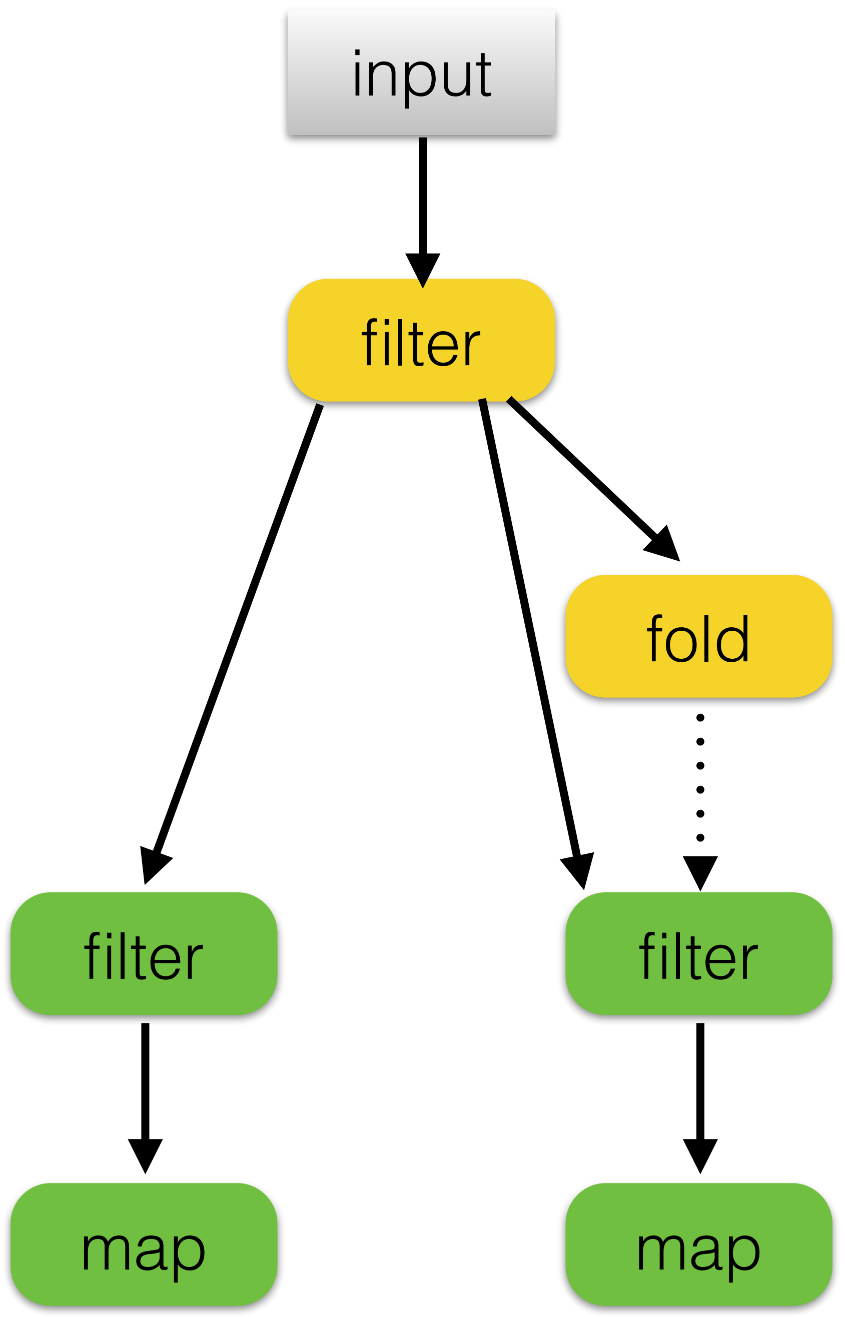
Paper published at FHPC'14:

Fusing filters with ILP

Finds "best" clustering of graph

Clustering

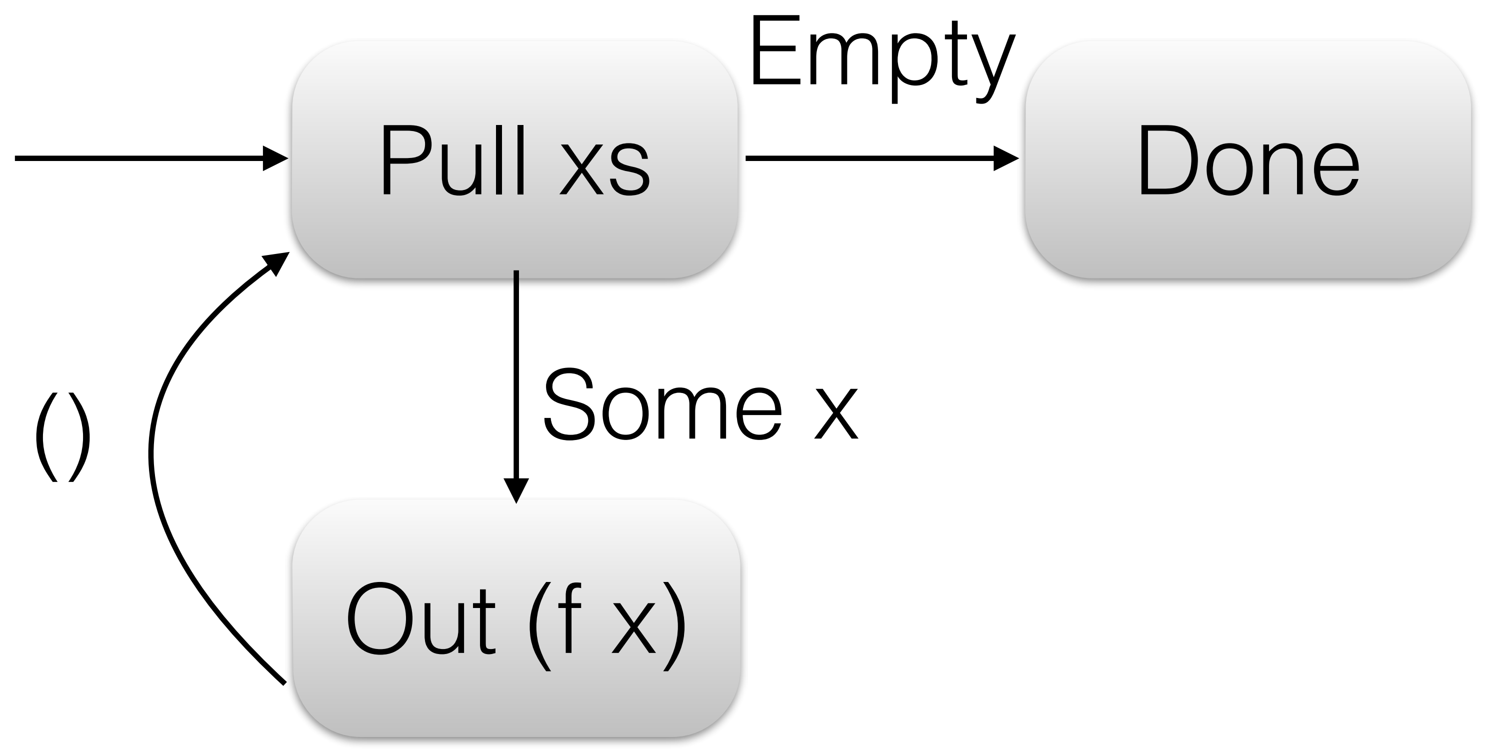
```
let ns    = filter  (>0)      input

    sum   = fold    (+) 0     ns
    ls    = filter  (>sum)    ns
    rs    = filter  odd       ns

    ls'   = map     (+1)      ls
    rs'   = map     (-1)      rs
in  …
```



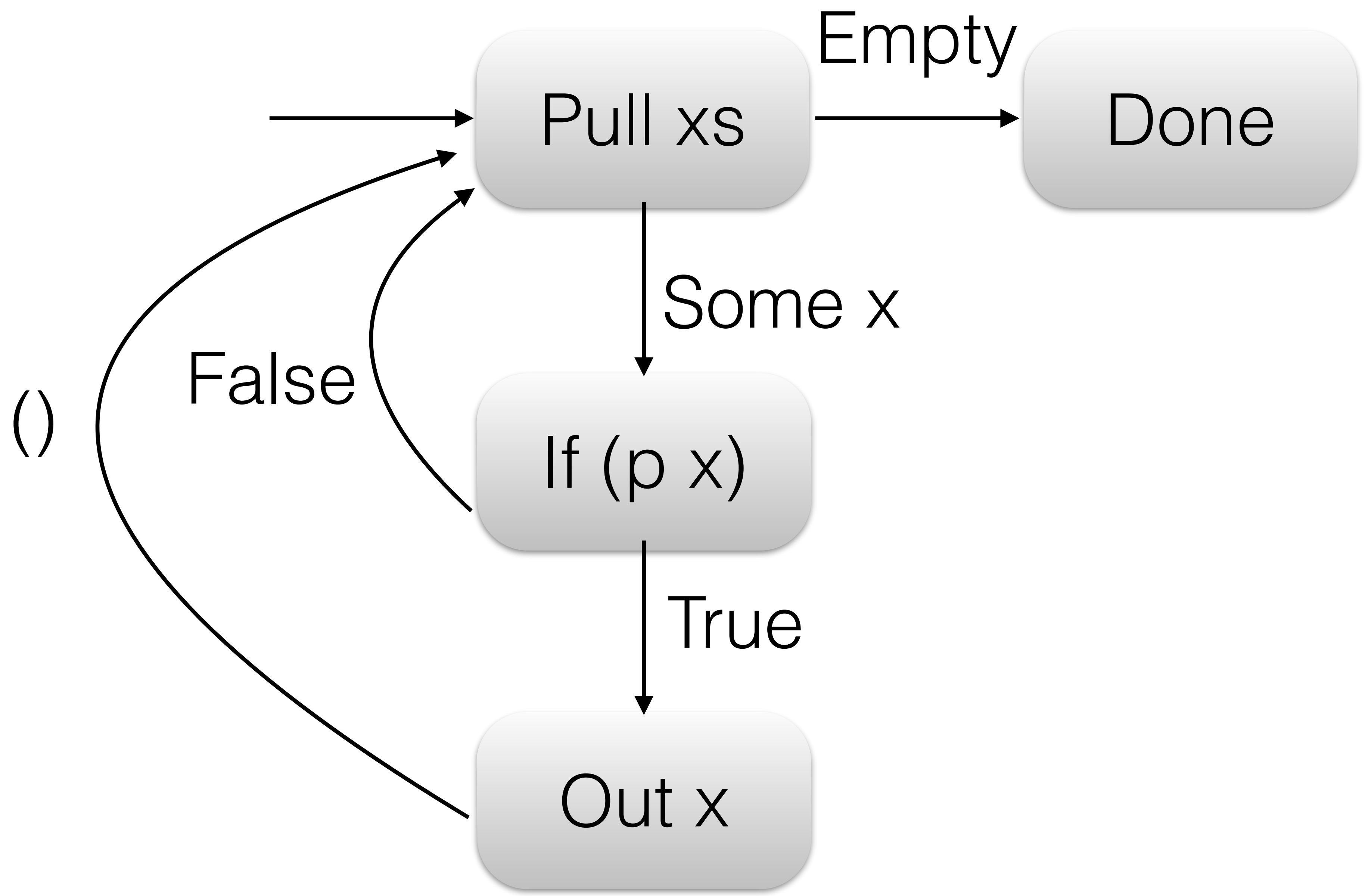
DFA fusion

(Image: Scott E. Harris, Wikipedia)

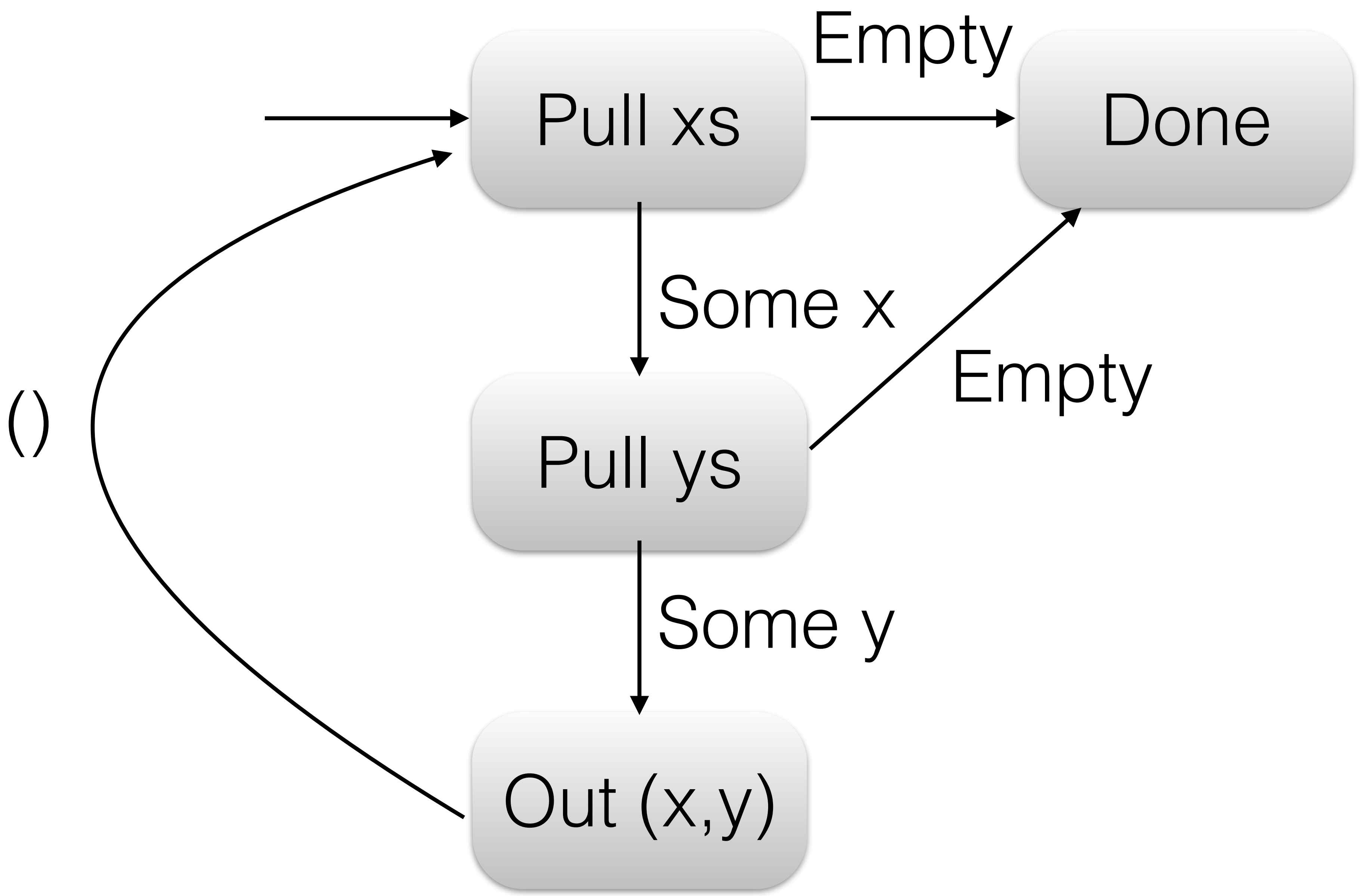
Map f xs



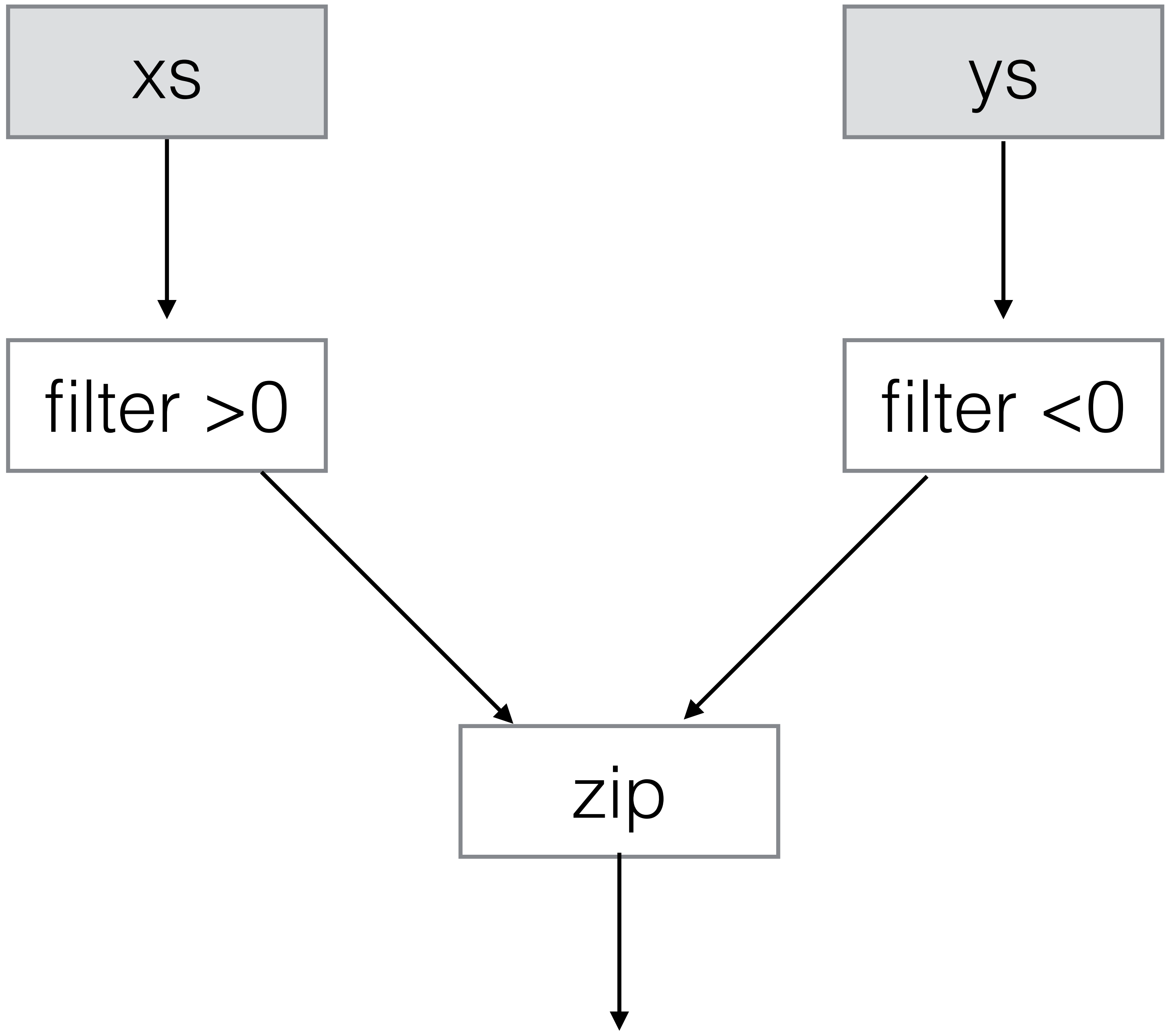
Filter p xs



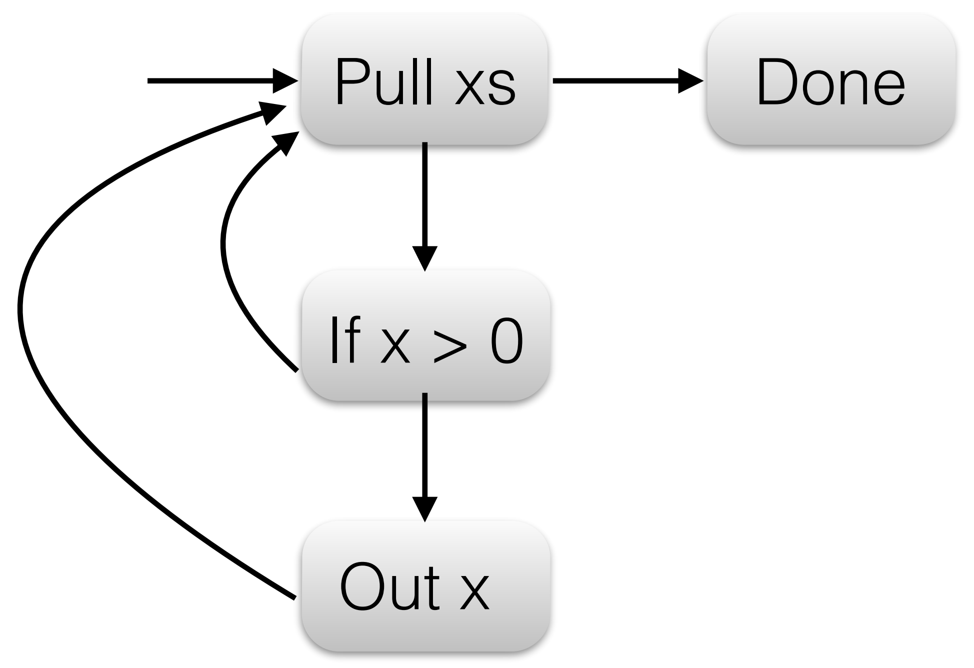
Zip xs ys



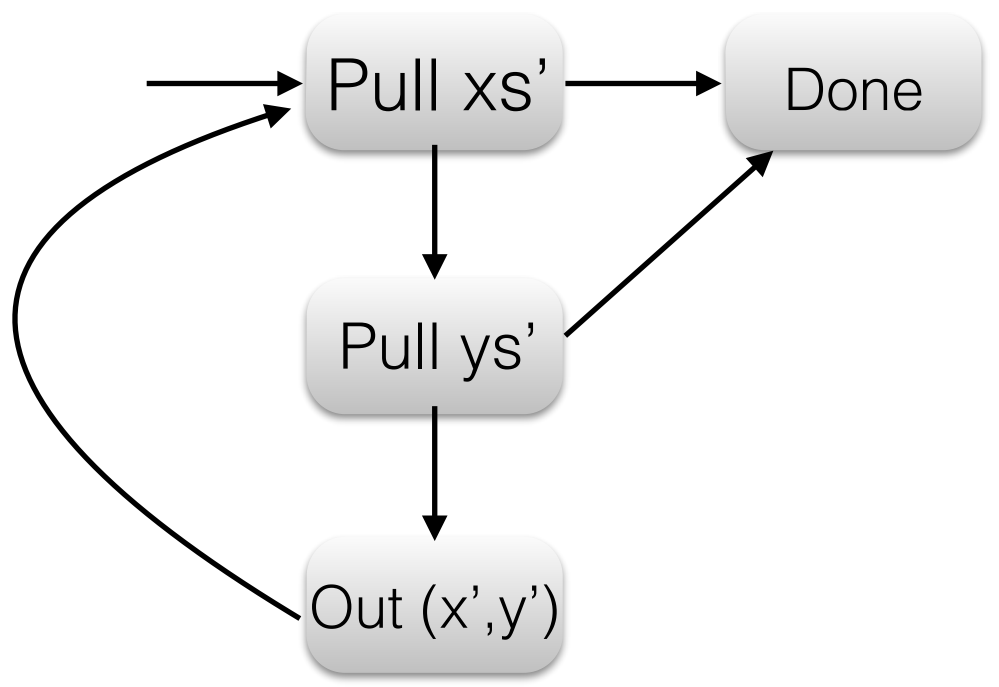
Pairing example



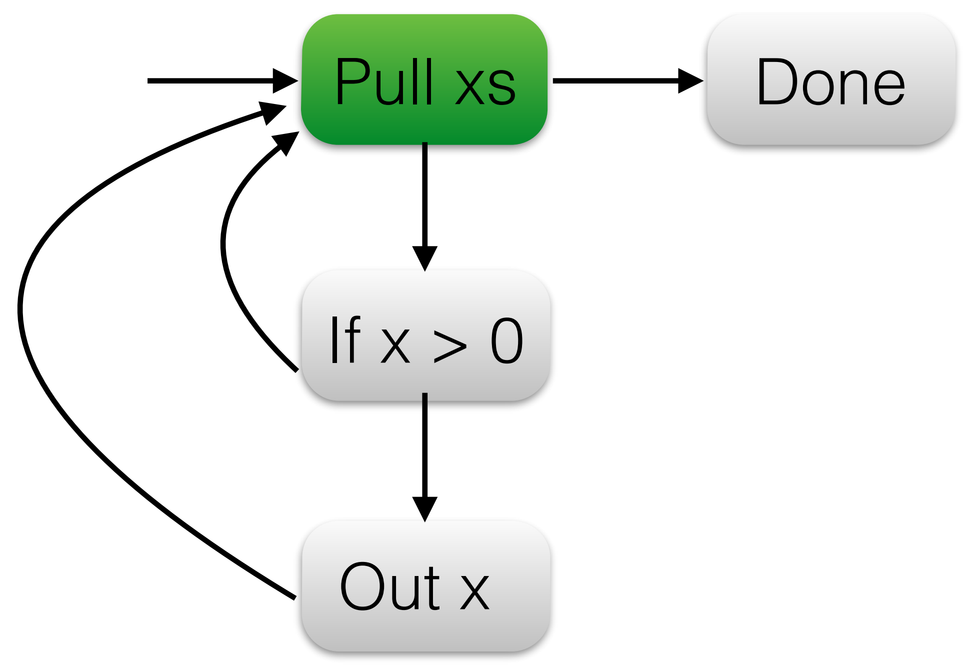
filter (>0) xs



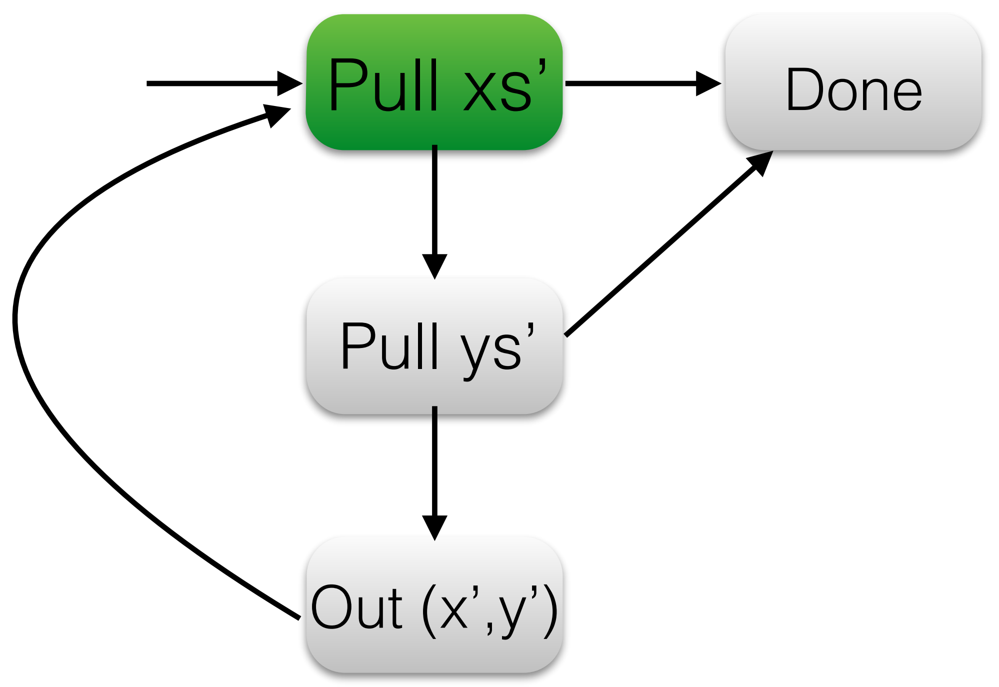
zip xs' ys'

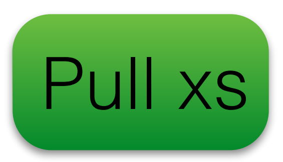


filter (>0) xs

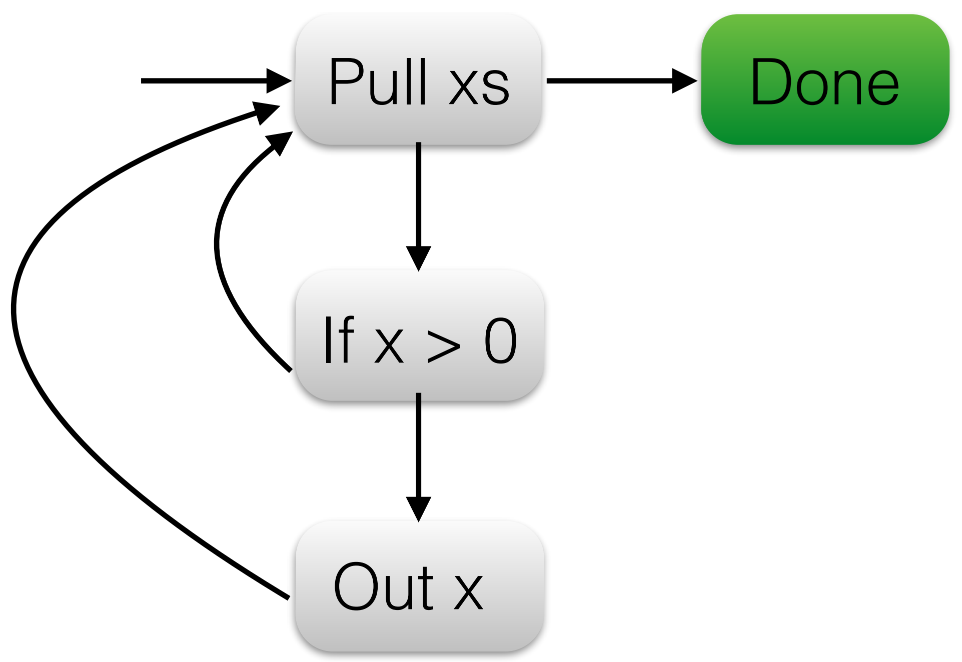


zip xs’ ys’

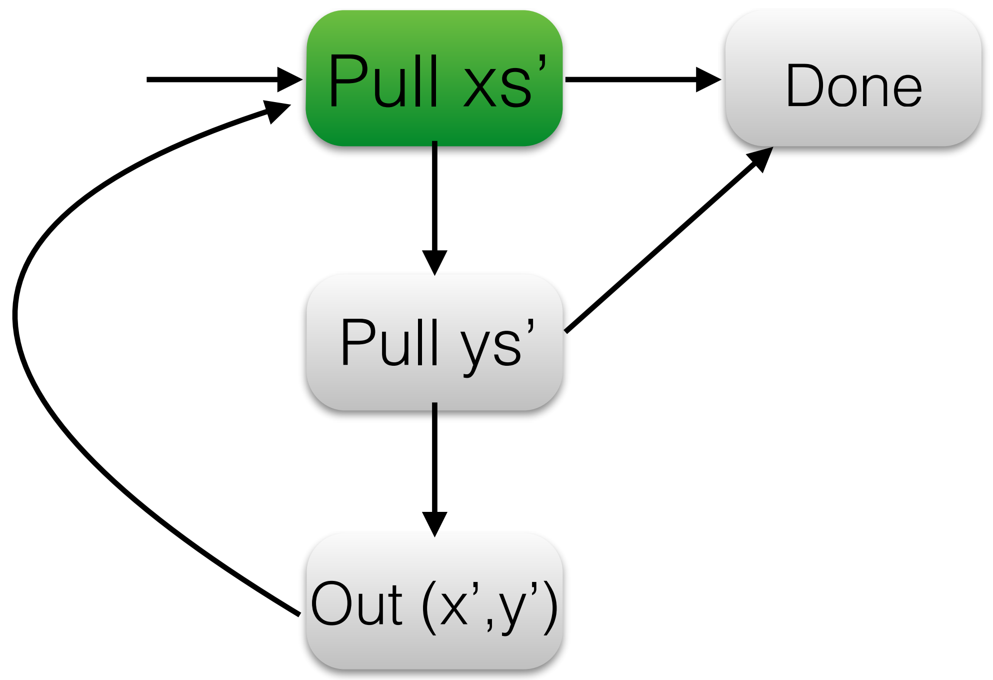


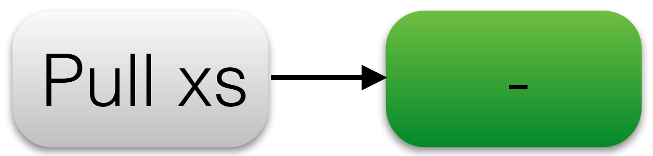


filter (>0) xs

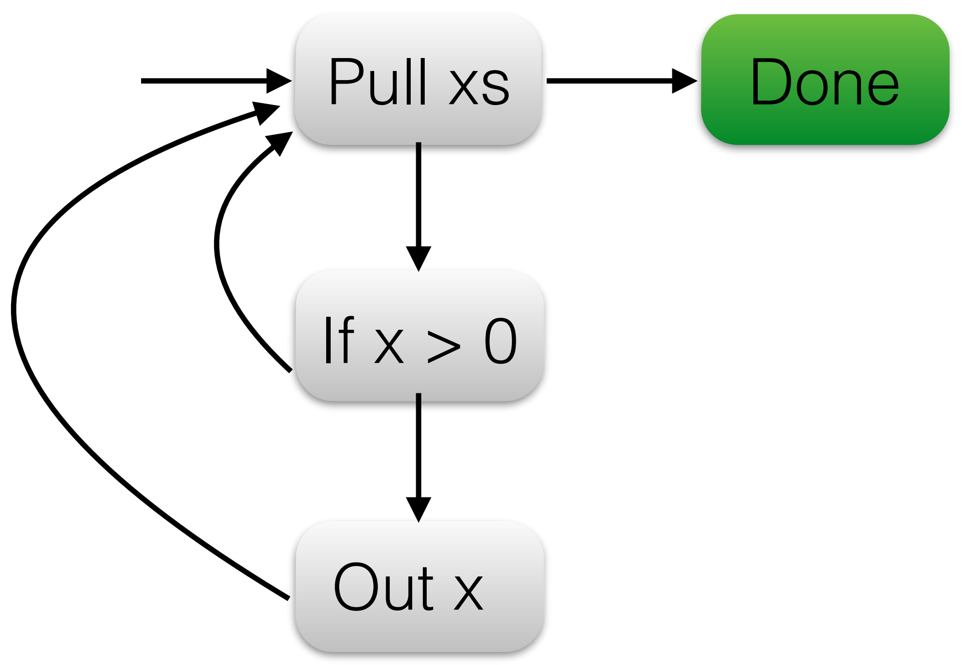


zip xs’ ys’

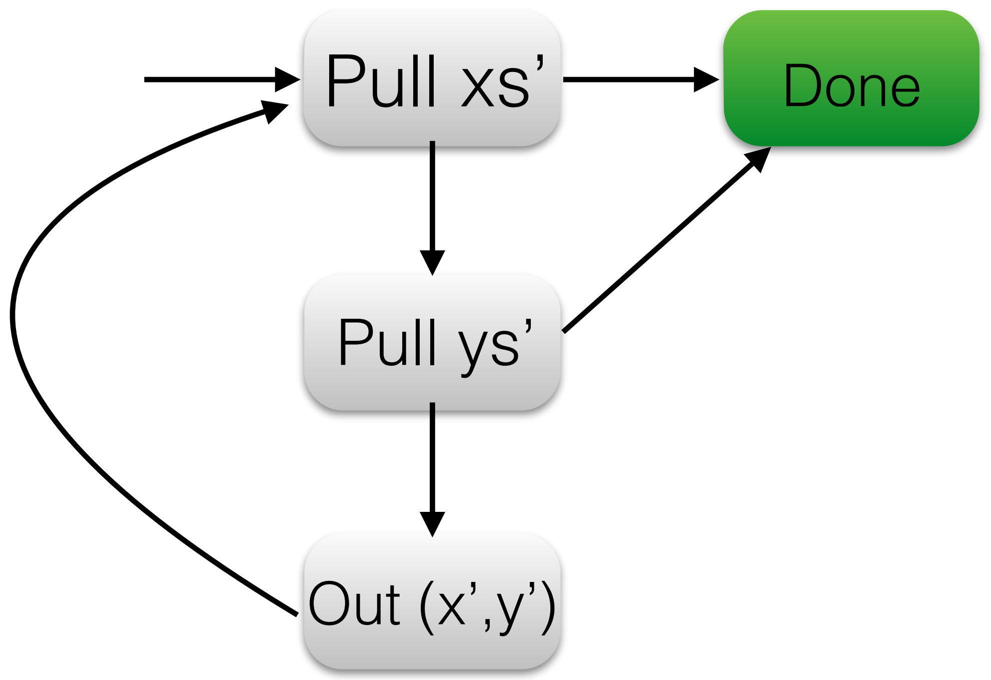


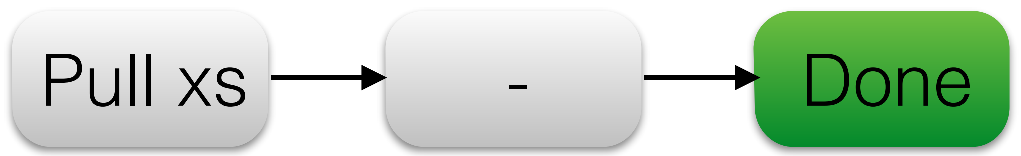


filter (>0) xs

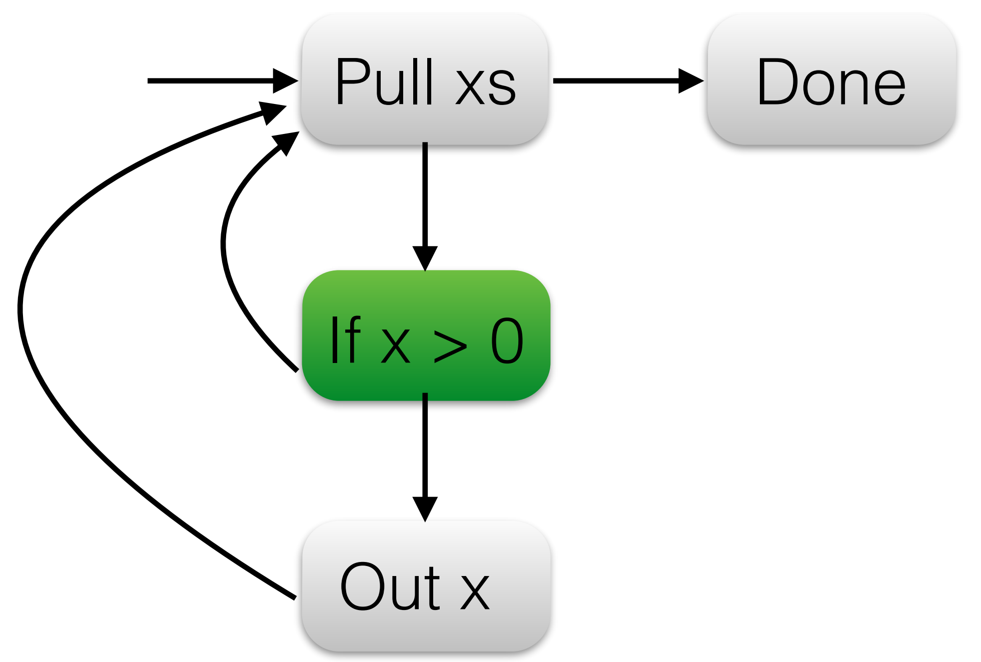


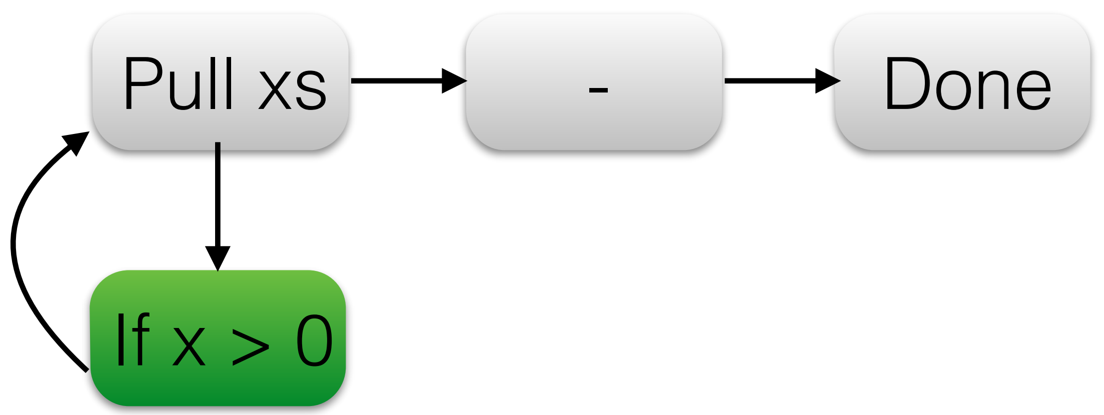
zip xs’ ys’



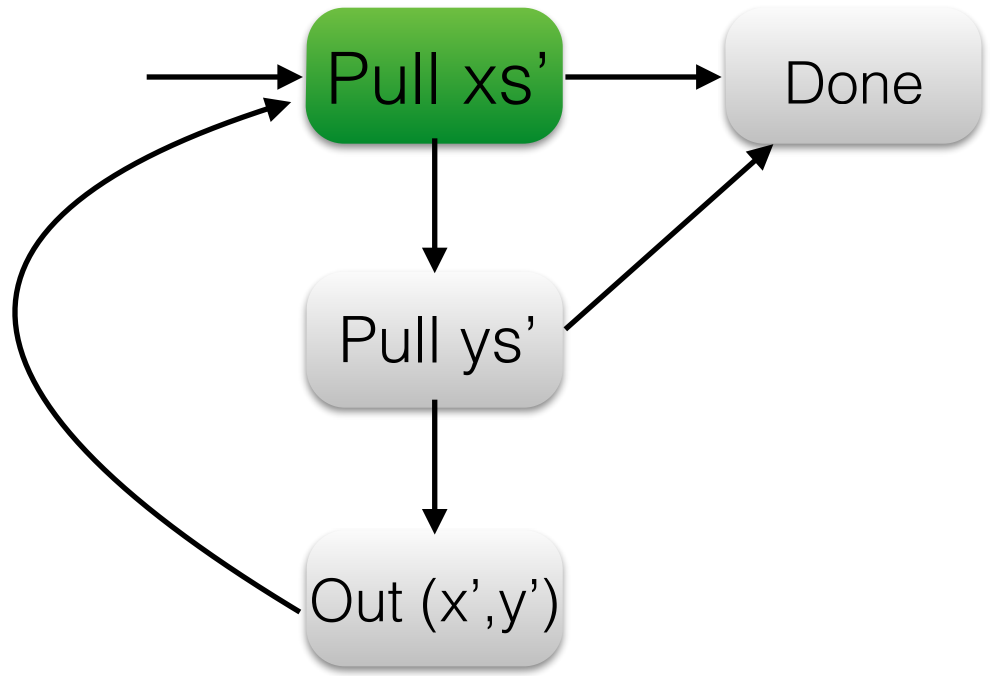


filter (>0) xs

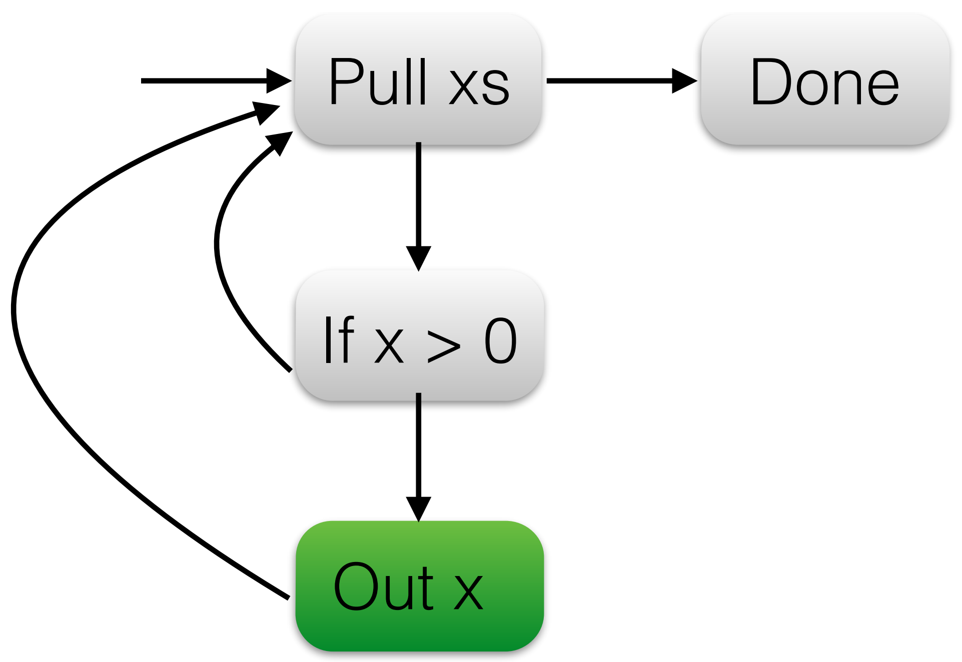


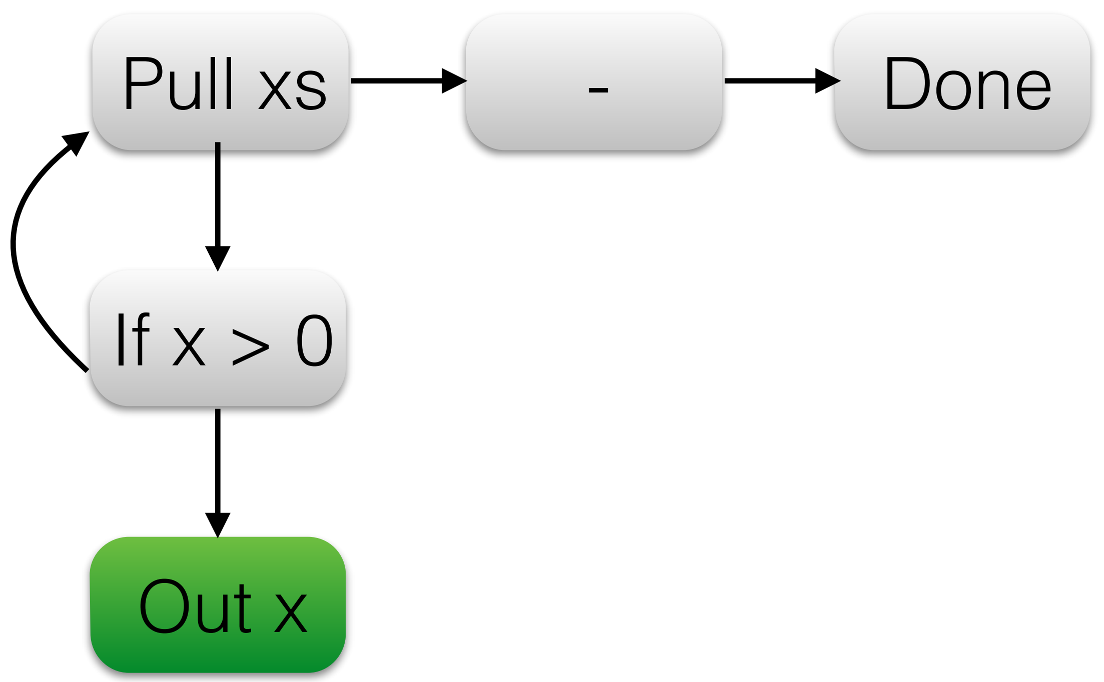


zip xs’ ys’

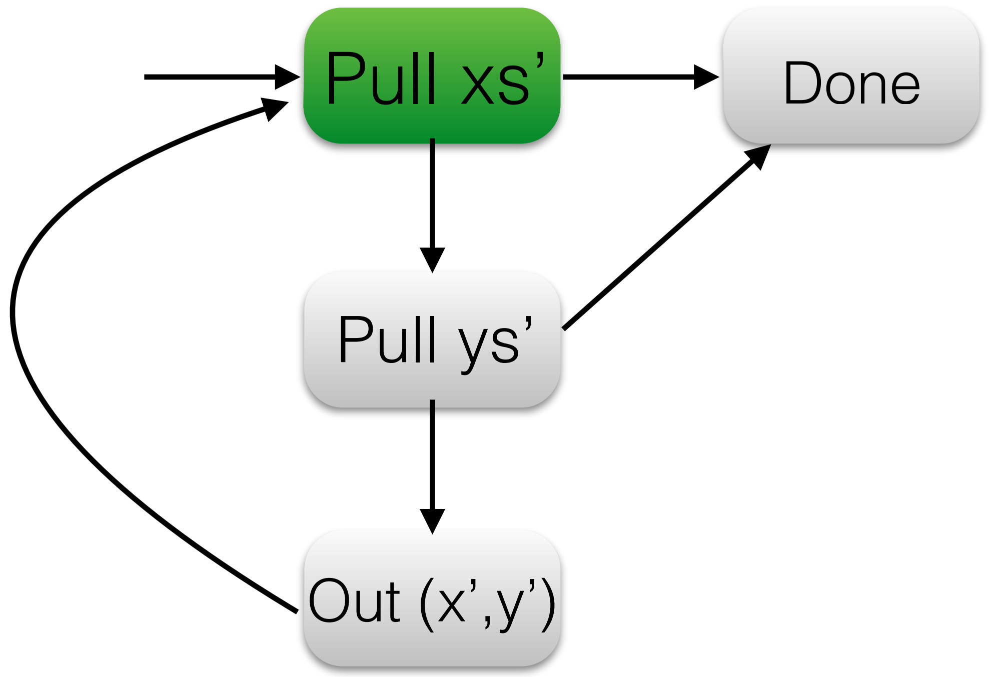


filter (>0) xs

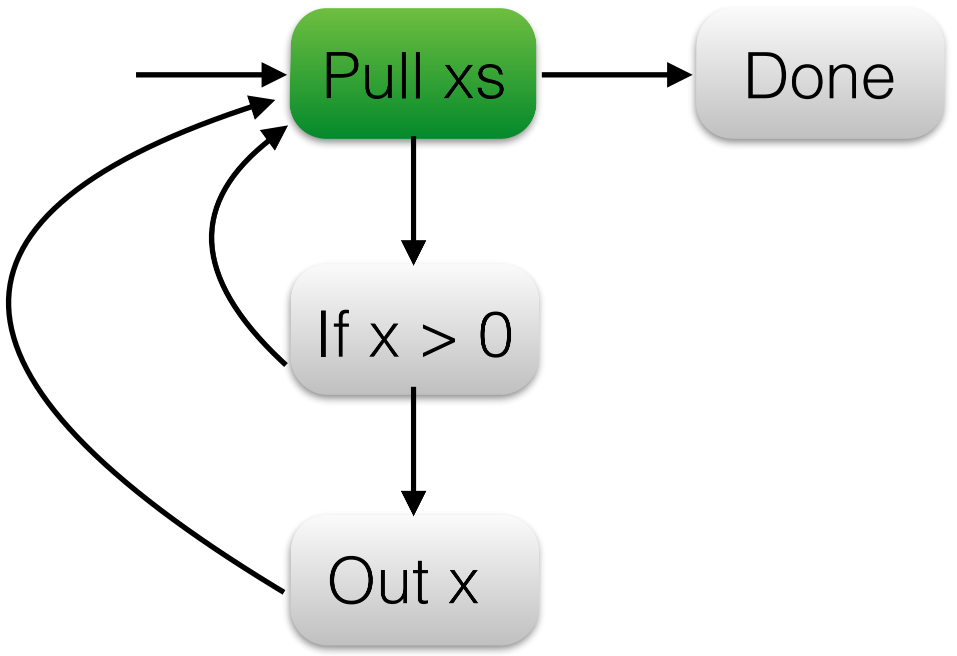




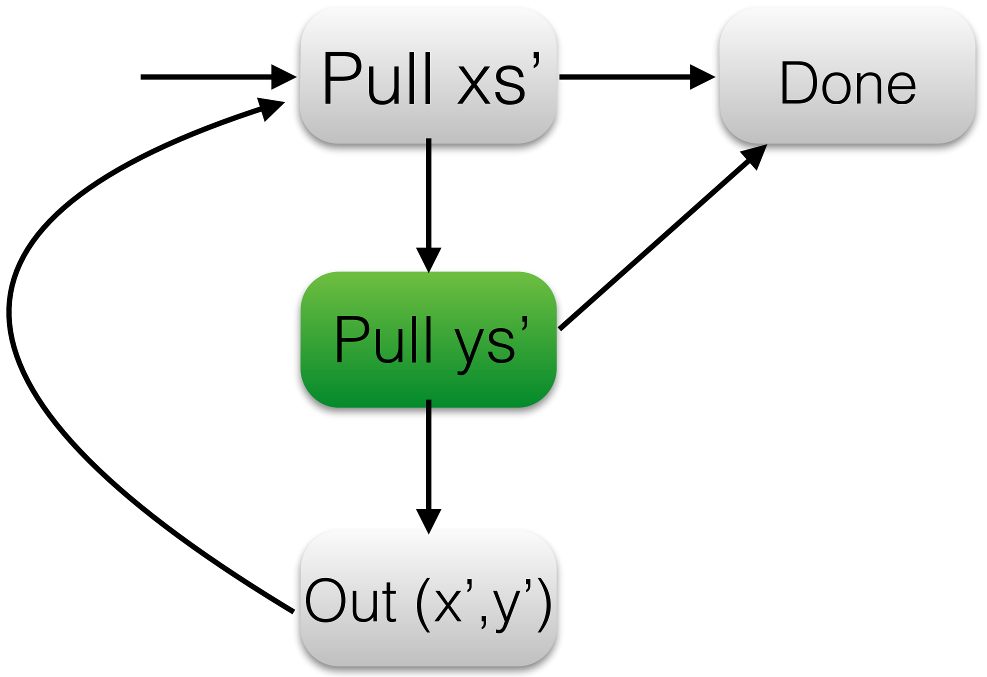
zip xs’ ys’

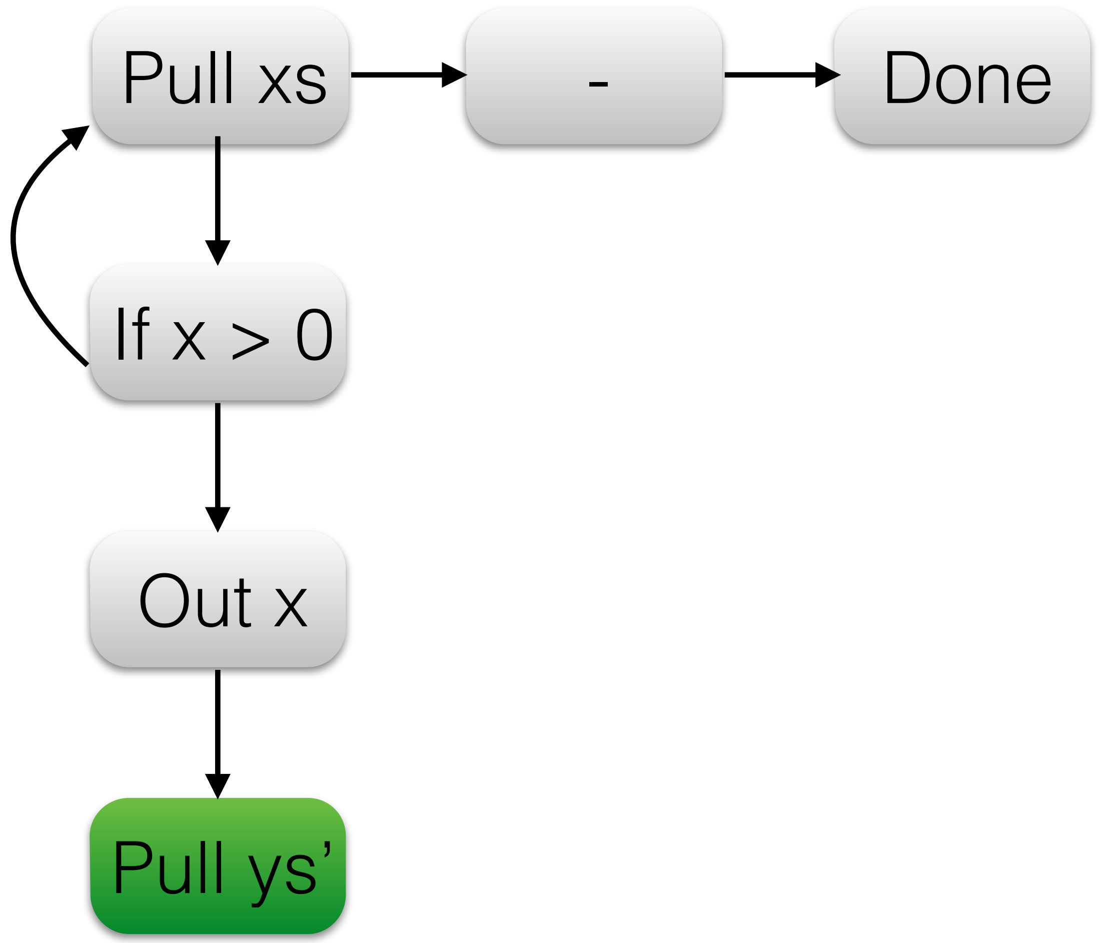


filter (>0) xs

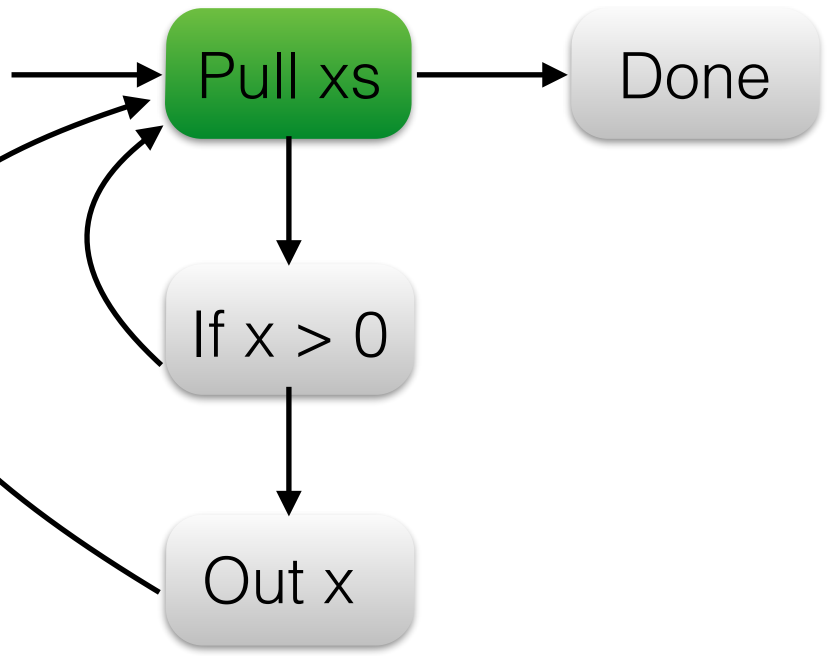


zip xs' ys'

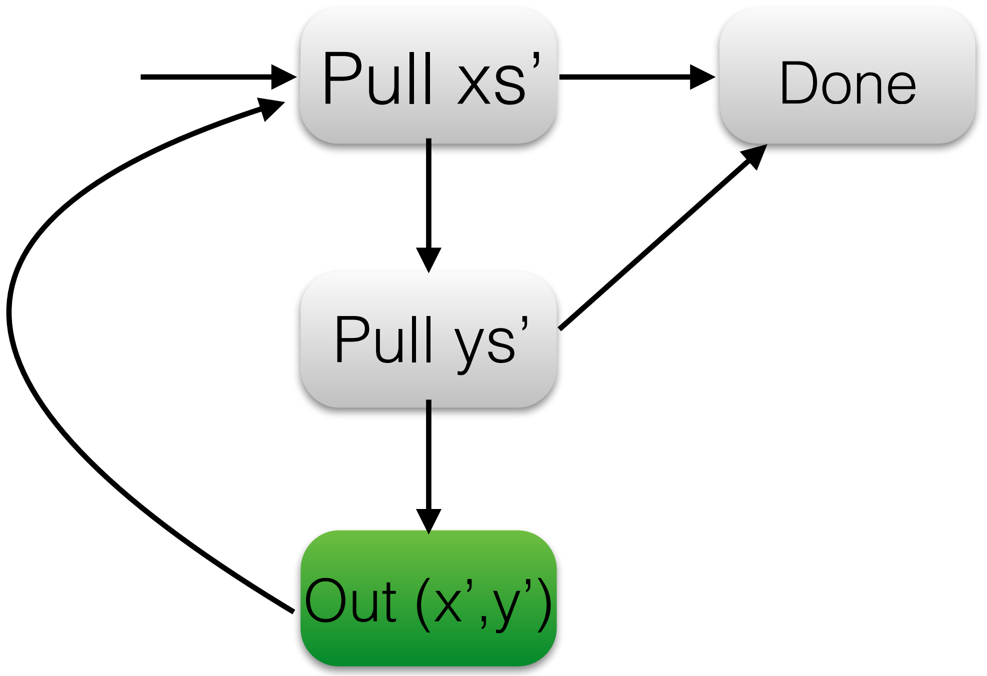


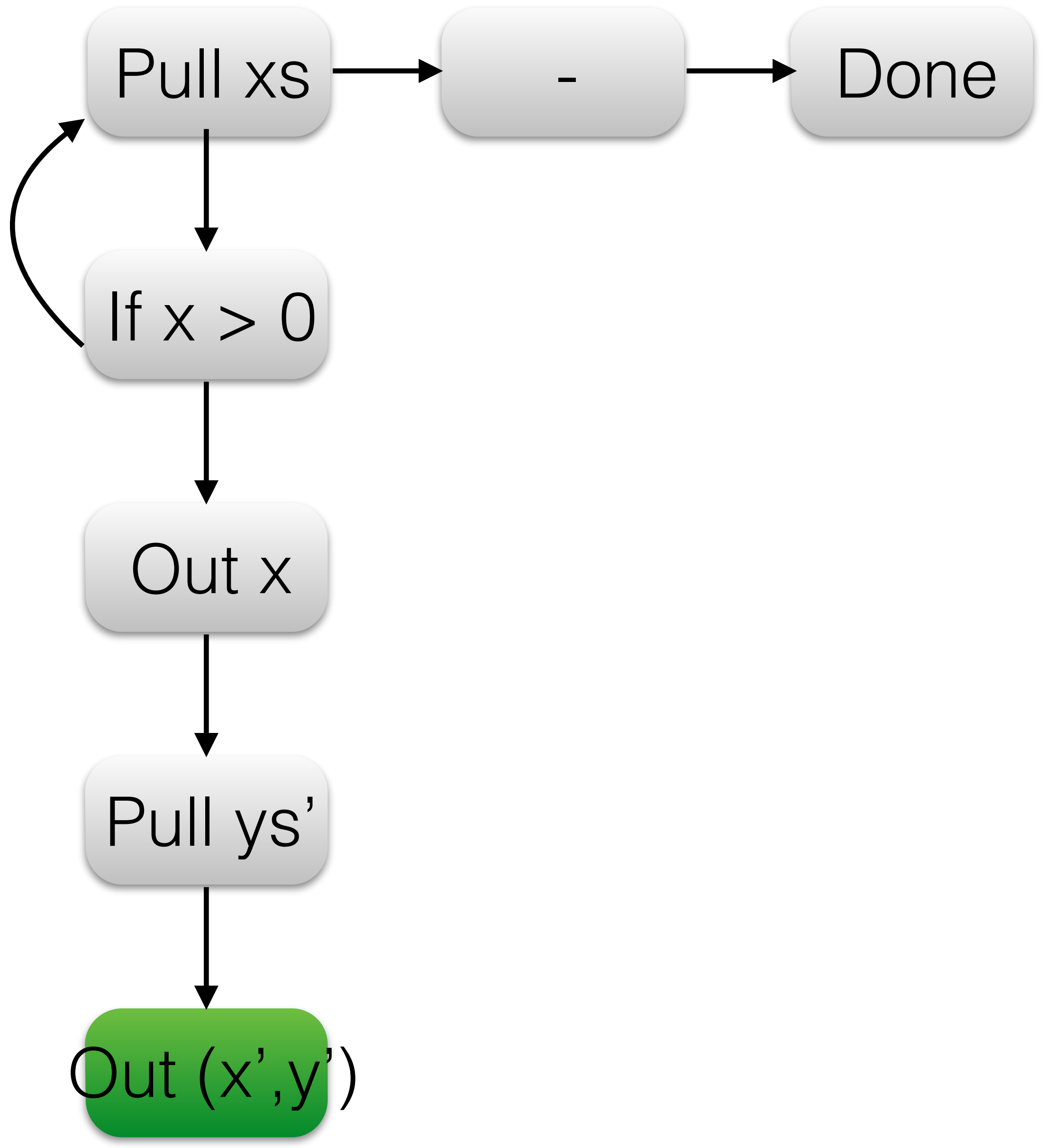


filter (>0) xs

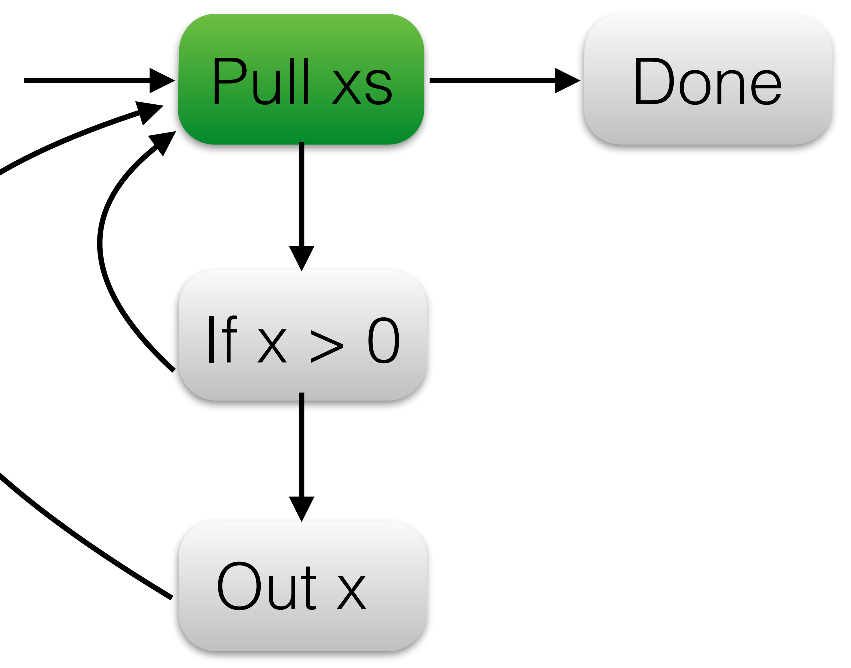


zip xs’ ys’

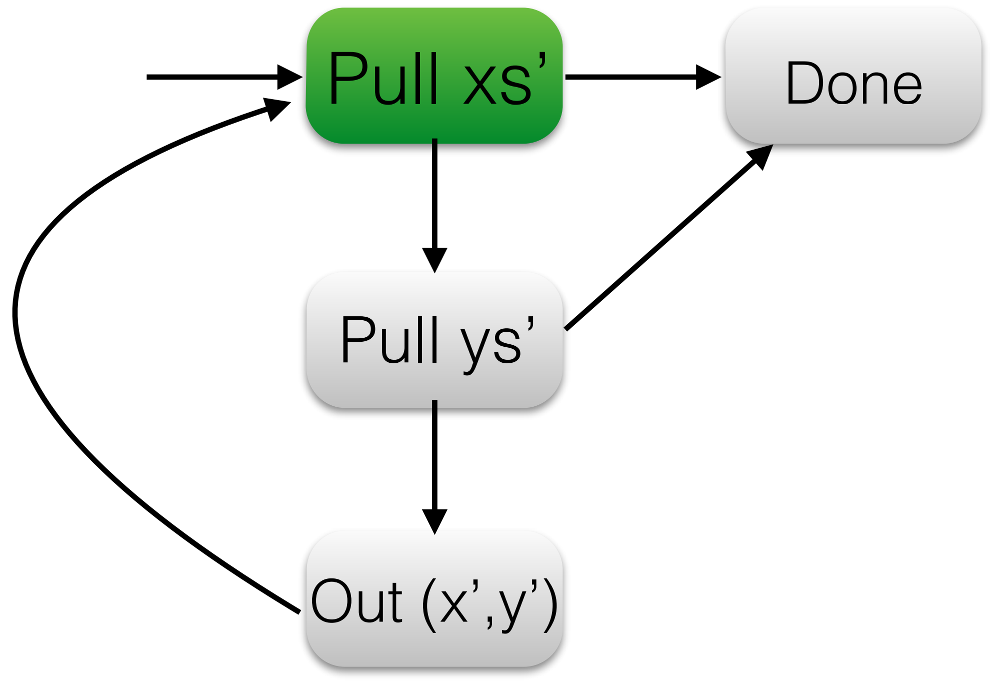


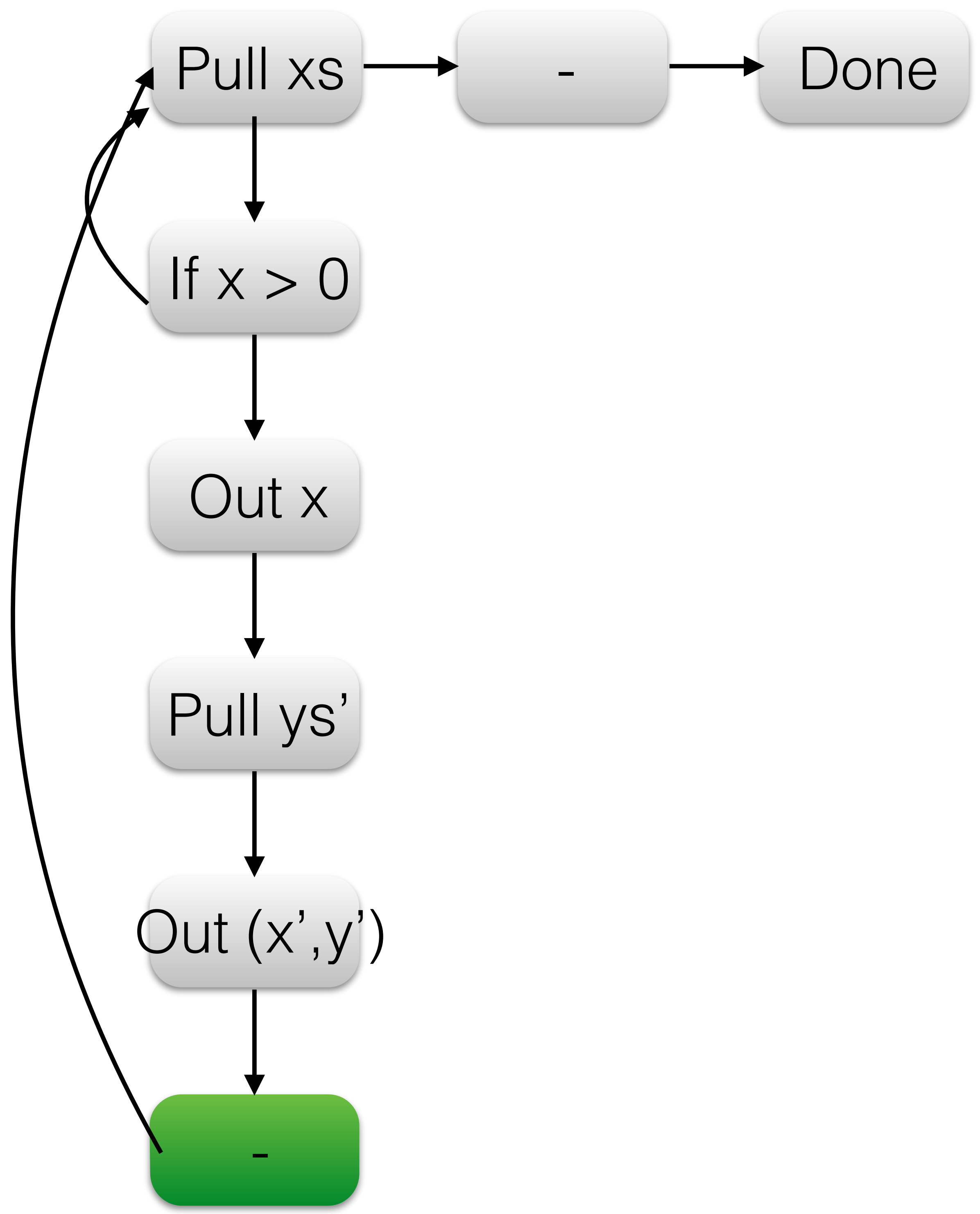


filter (>0) xs

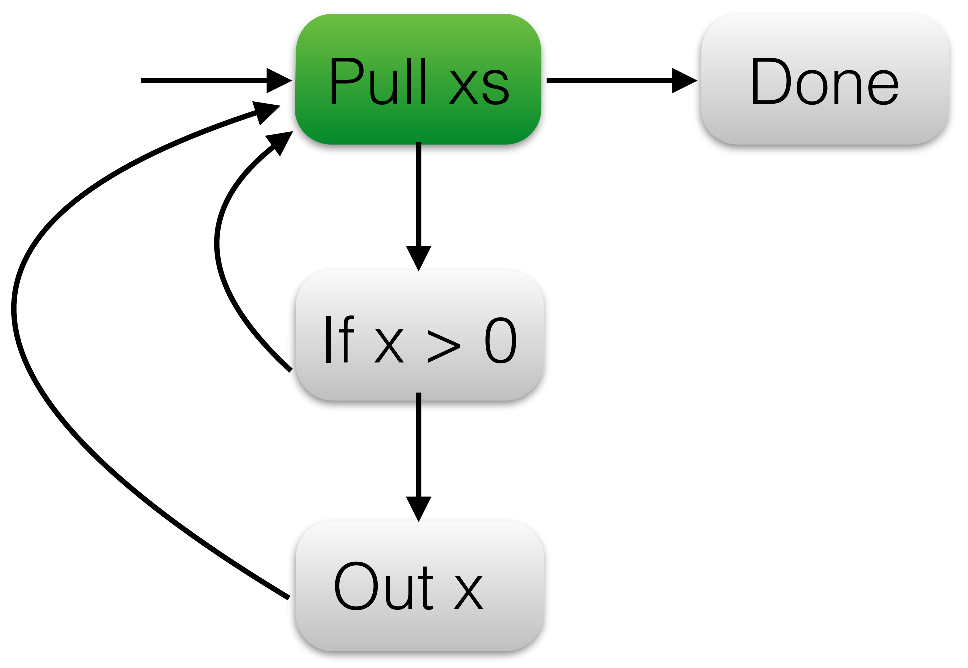


zip xs' ys'

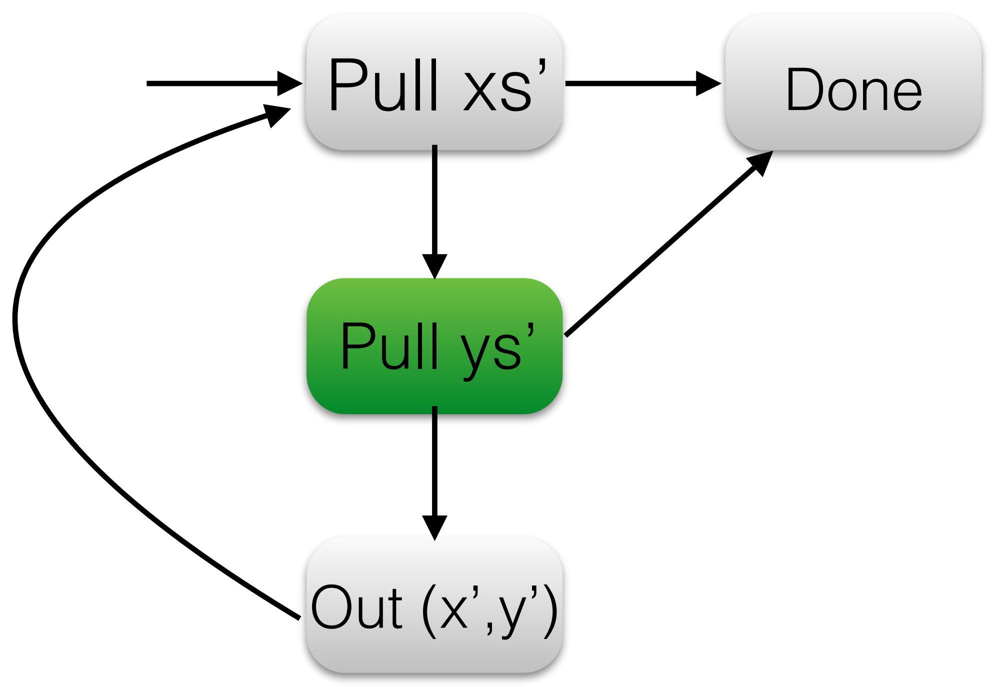


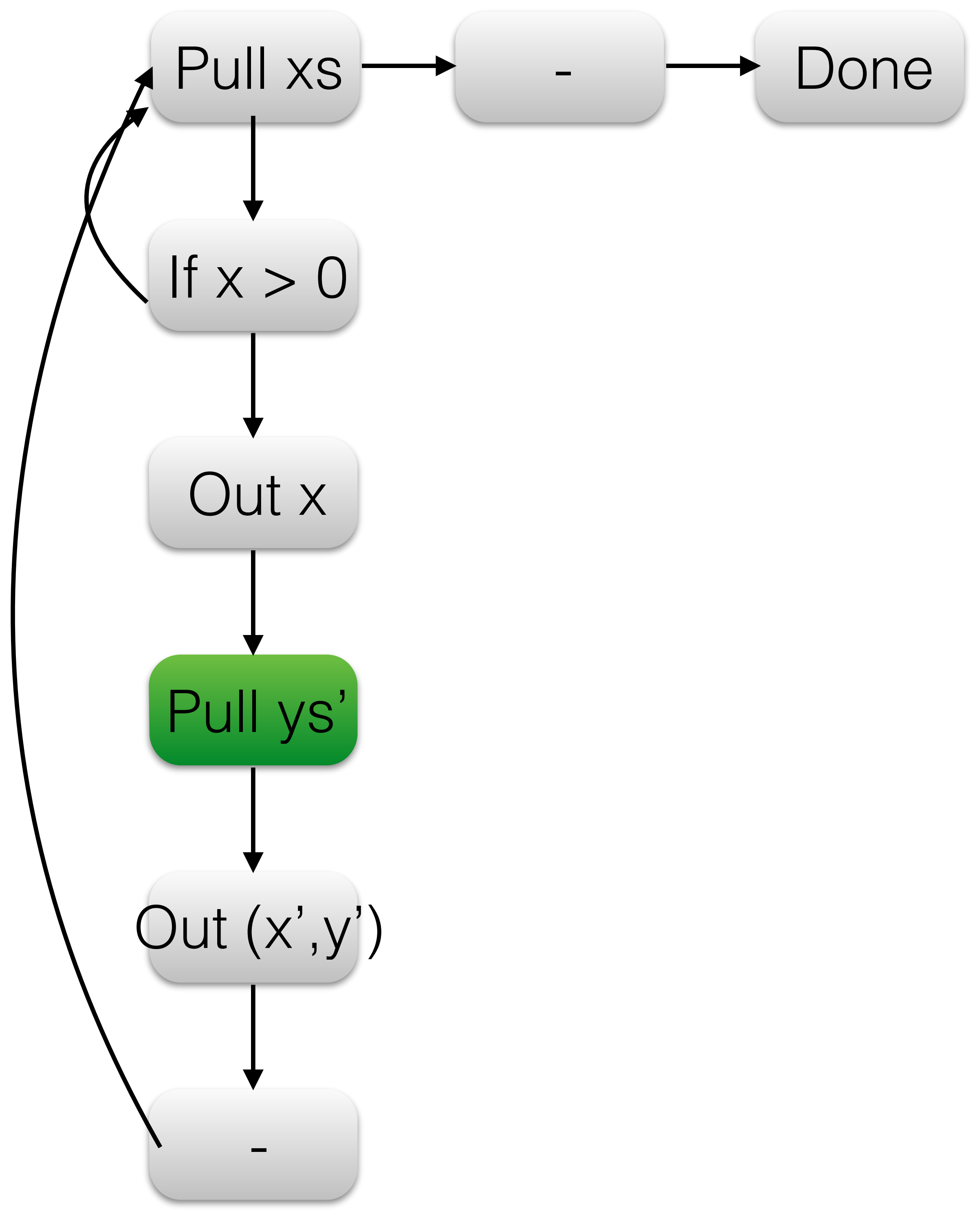


filter (>0) xs

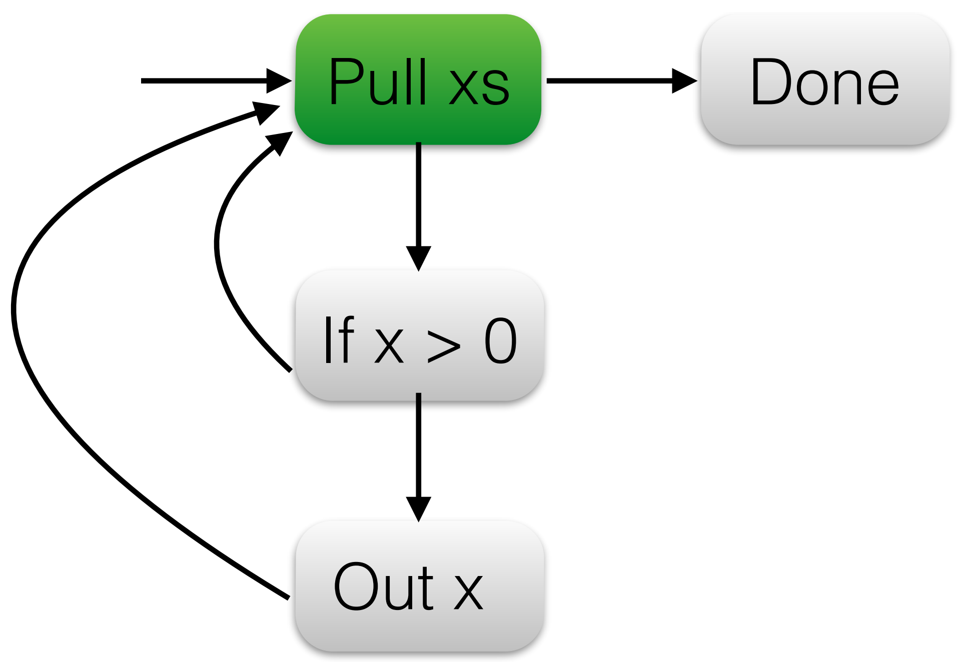


zip xs’ ys’

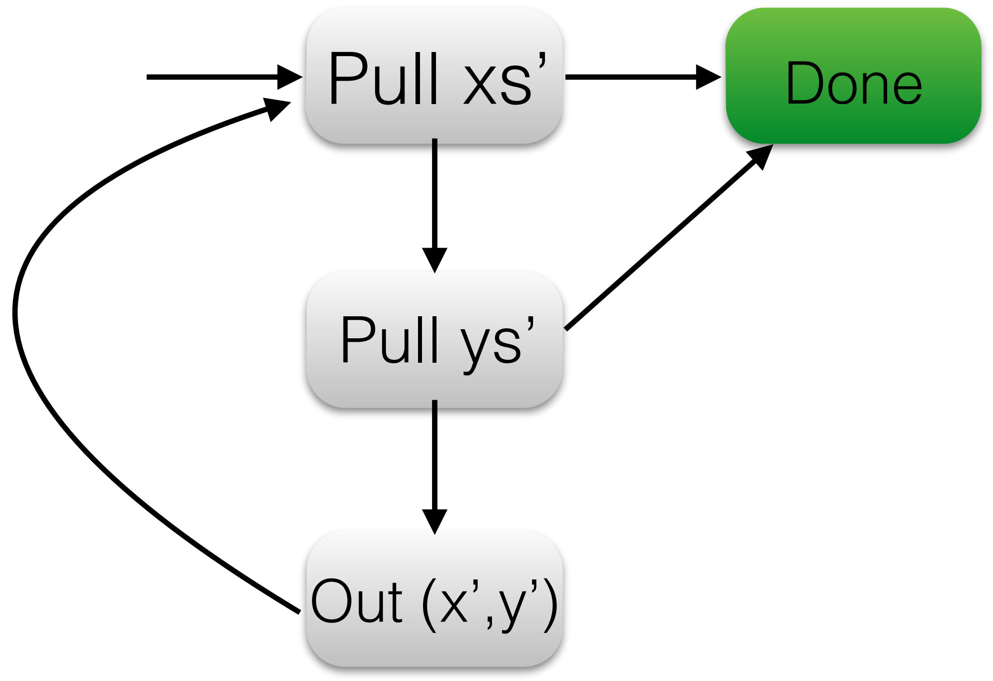


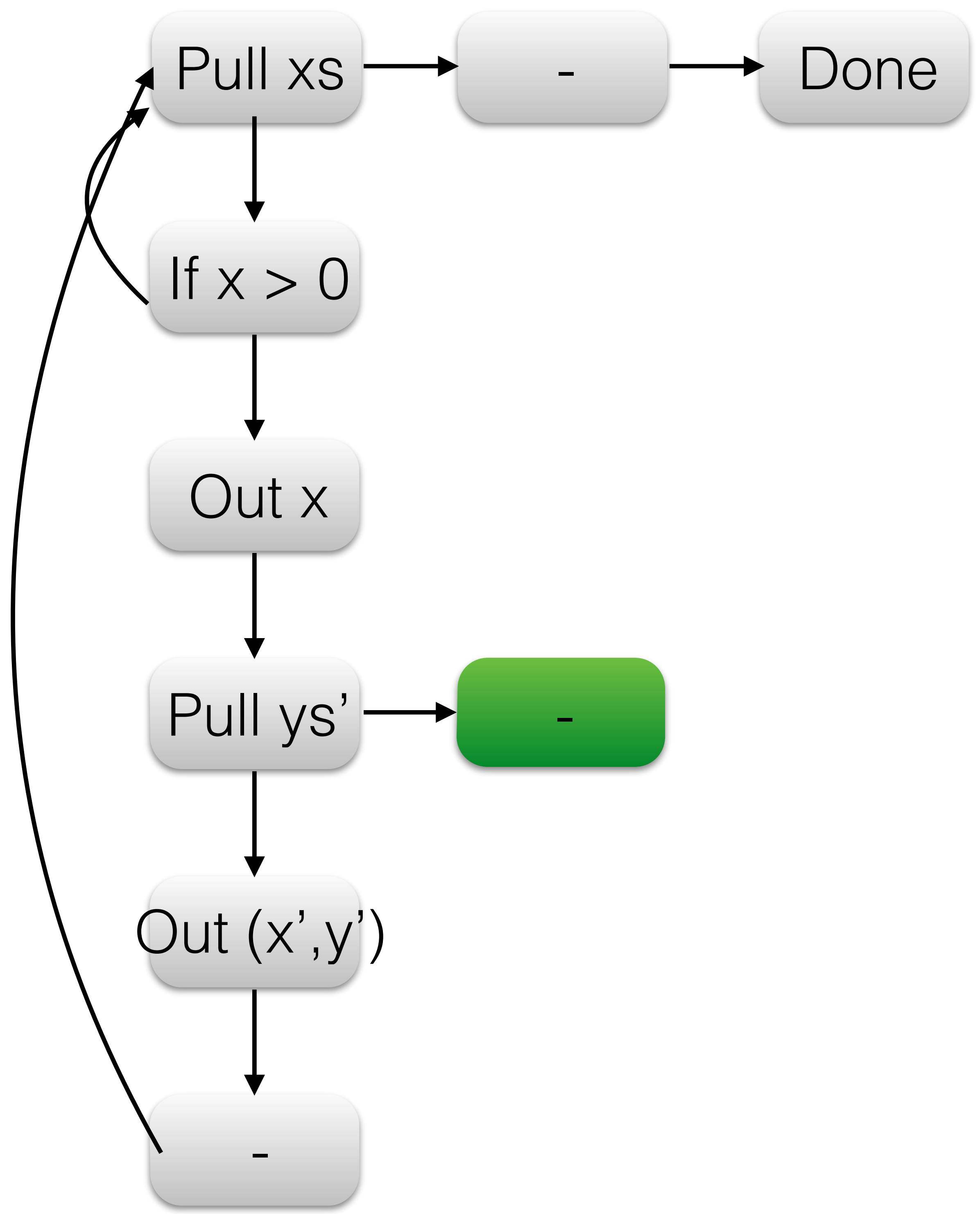


filter (>0) xs

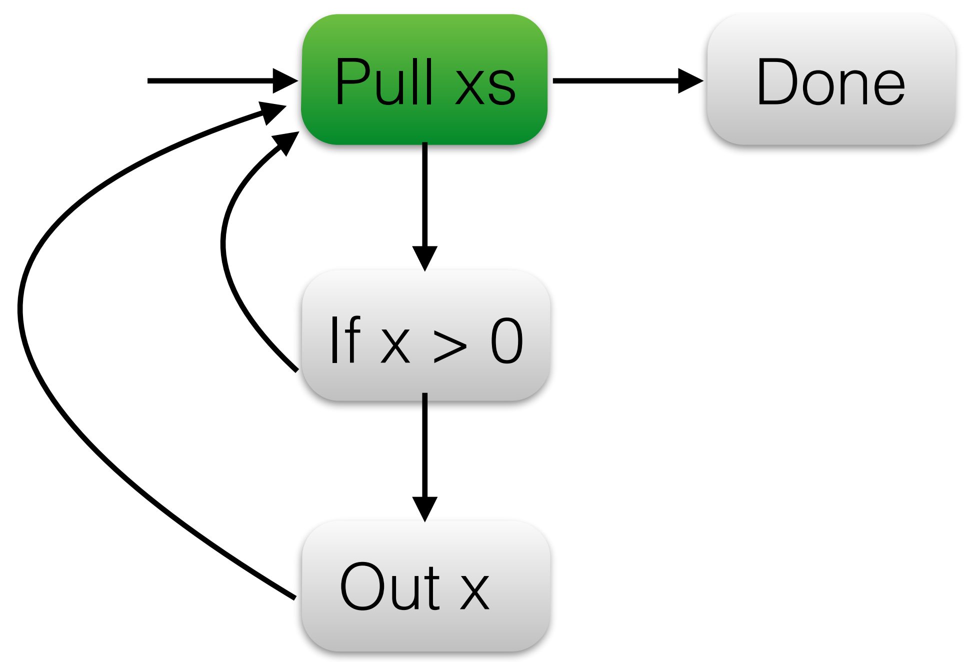


zip xs' ys'

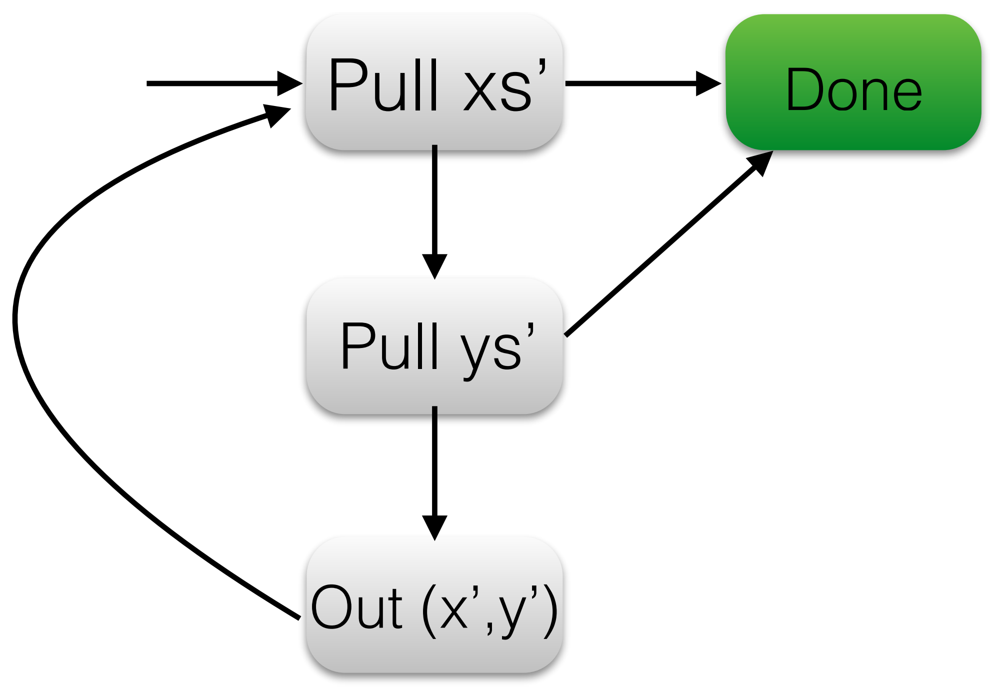


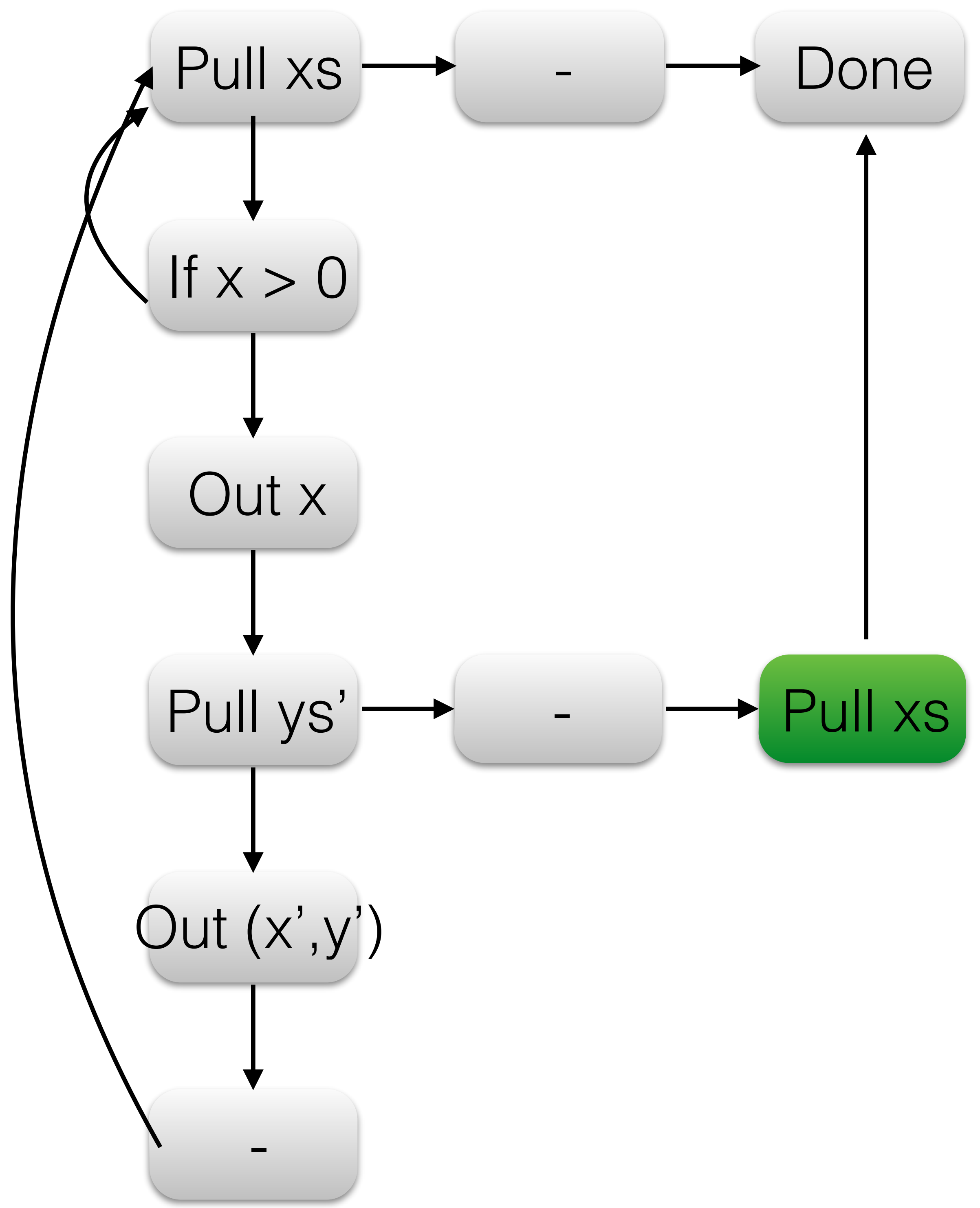


filter (>0) xs

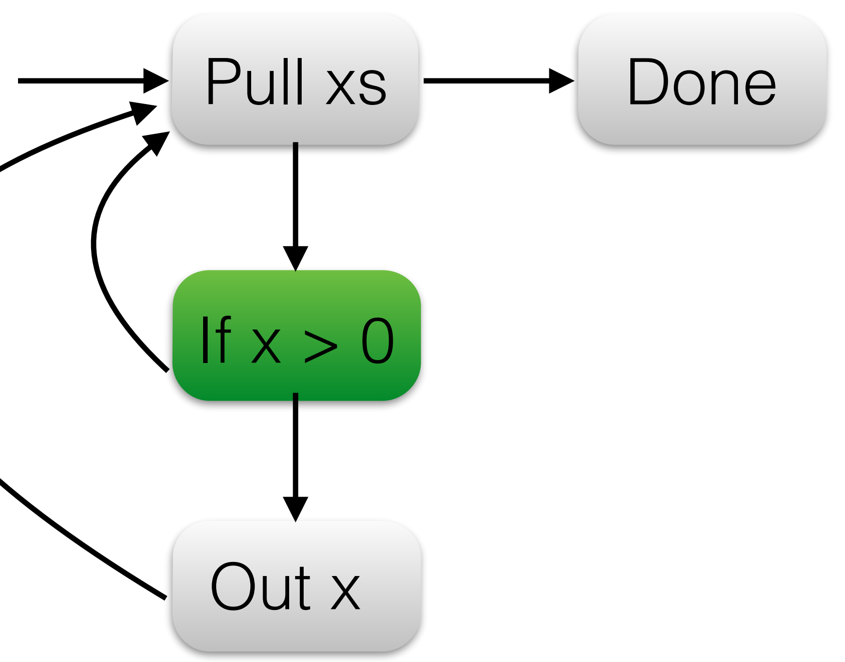


zip xs’ ys’

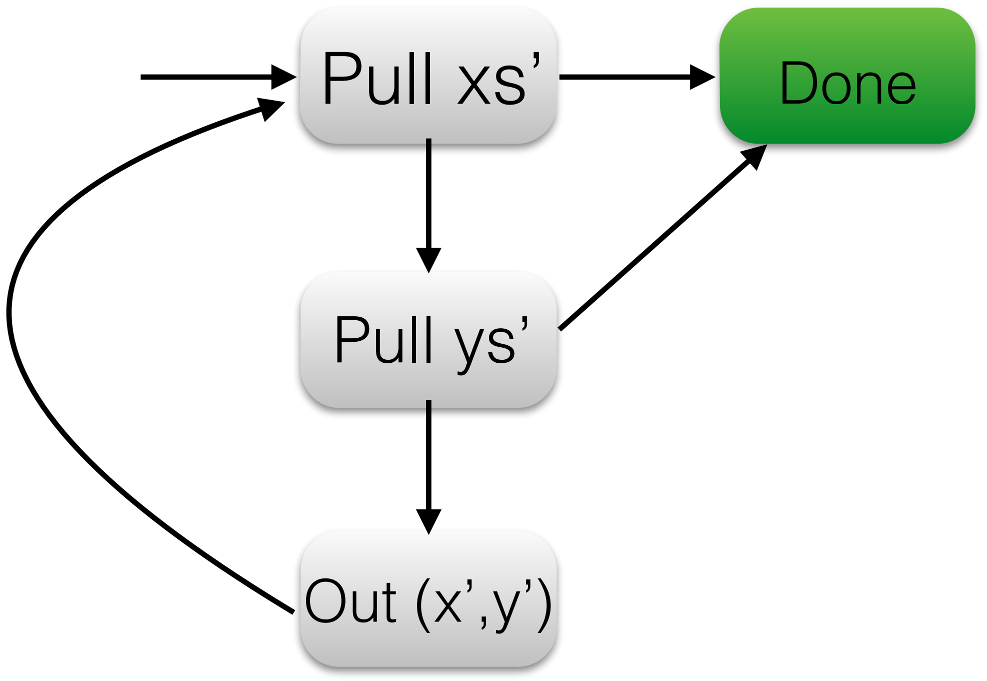


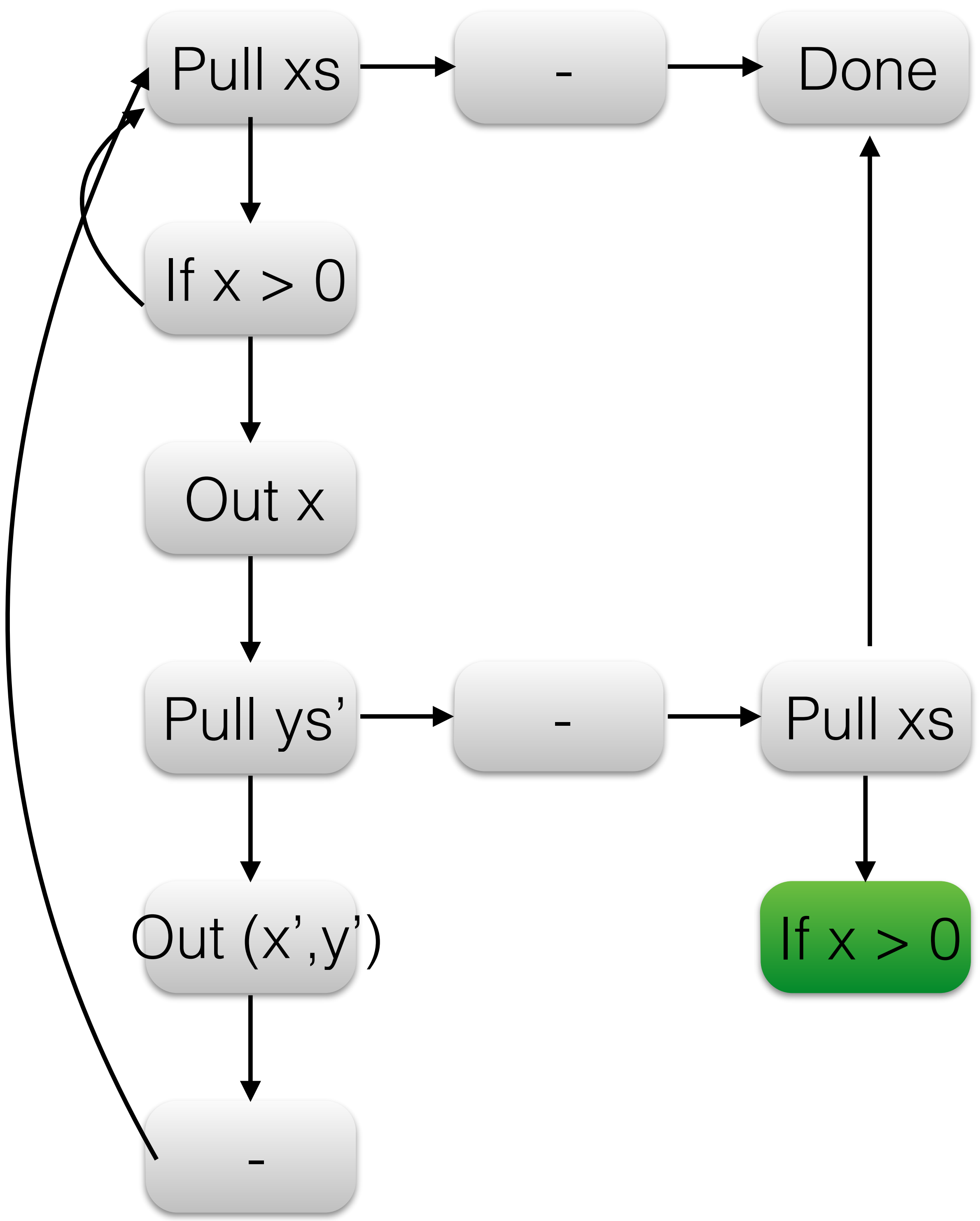


filter (>0) xs

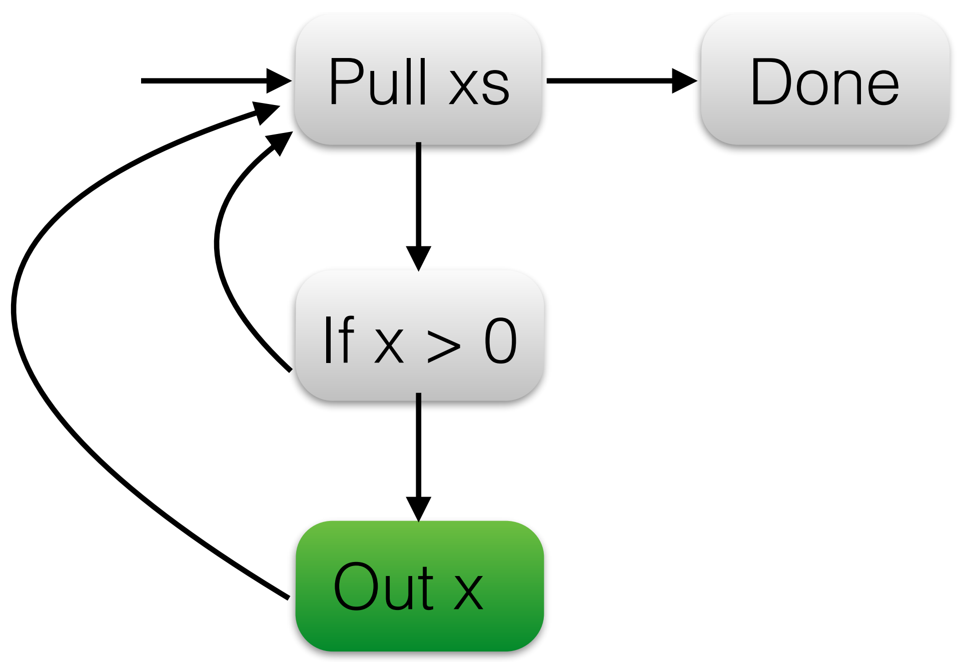


zip xs’ ys’

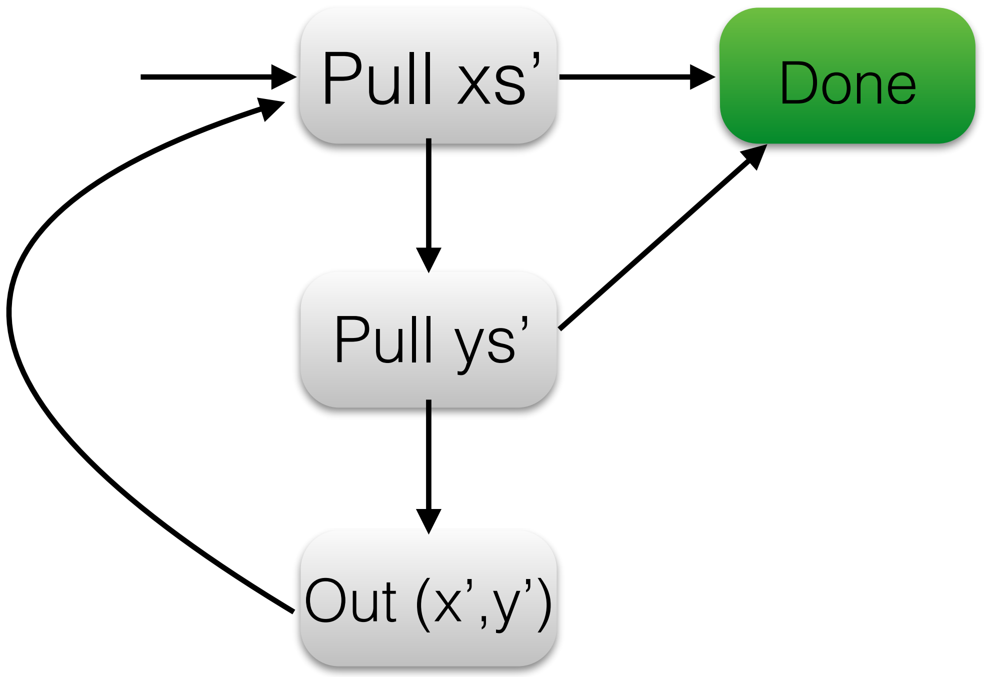


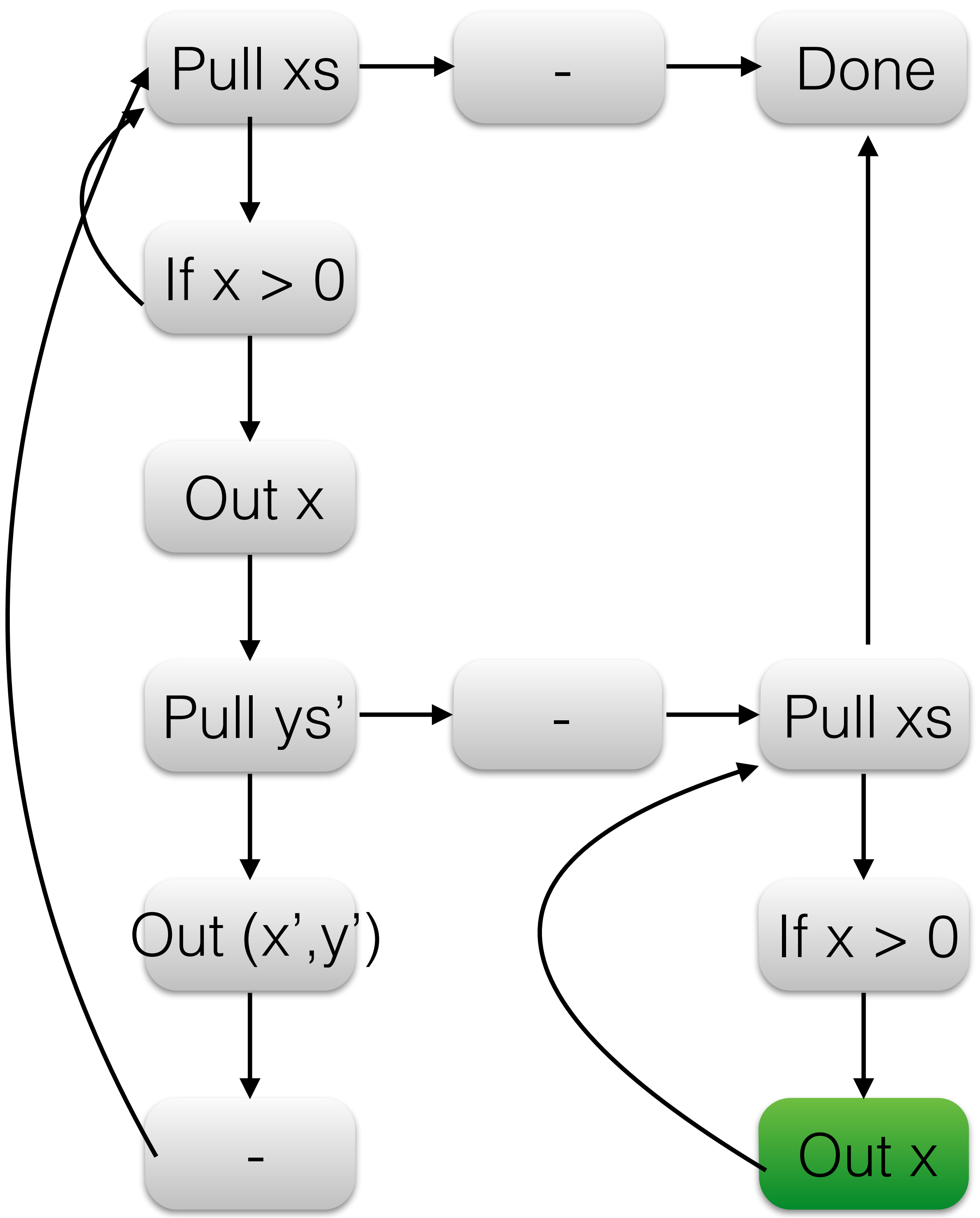


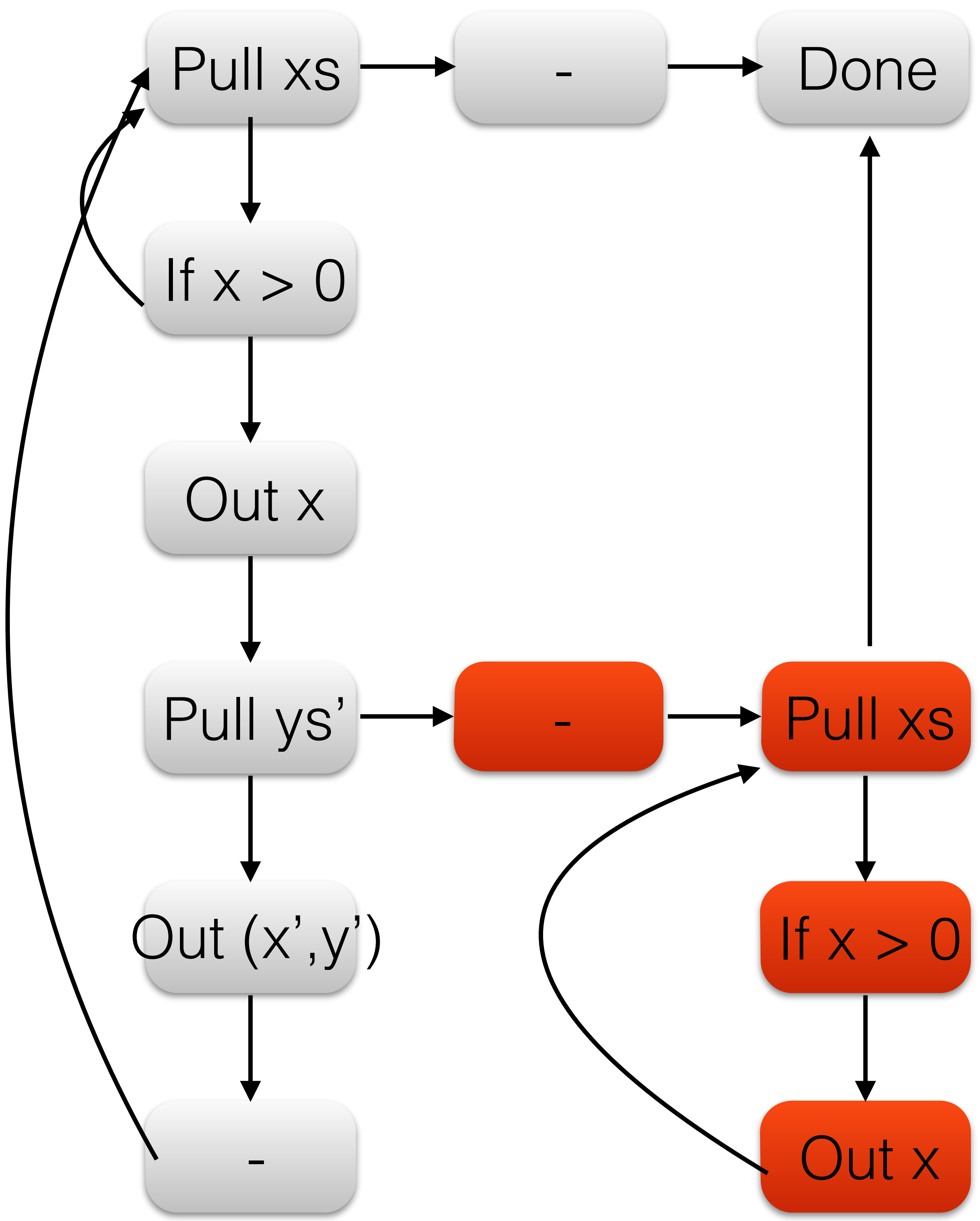
filter (>0) xs



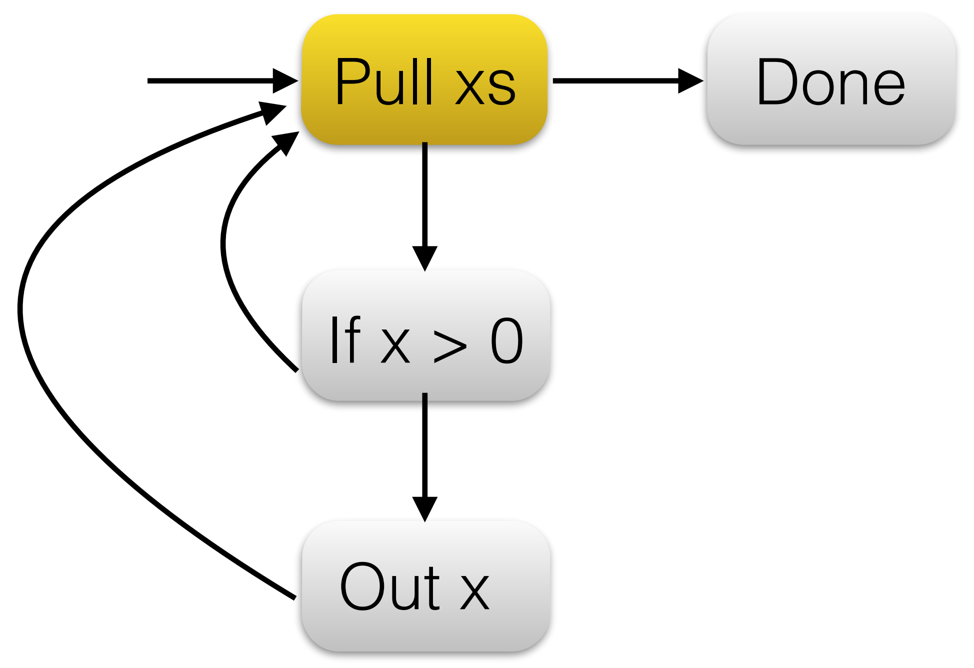
zip xs' ys'



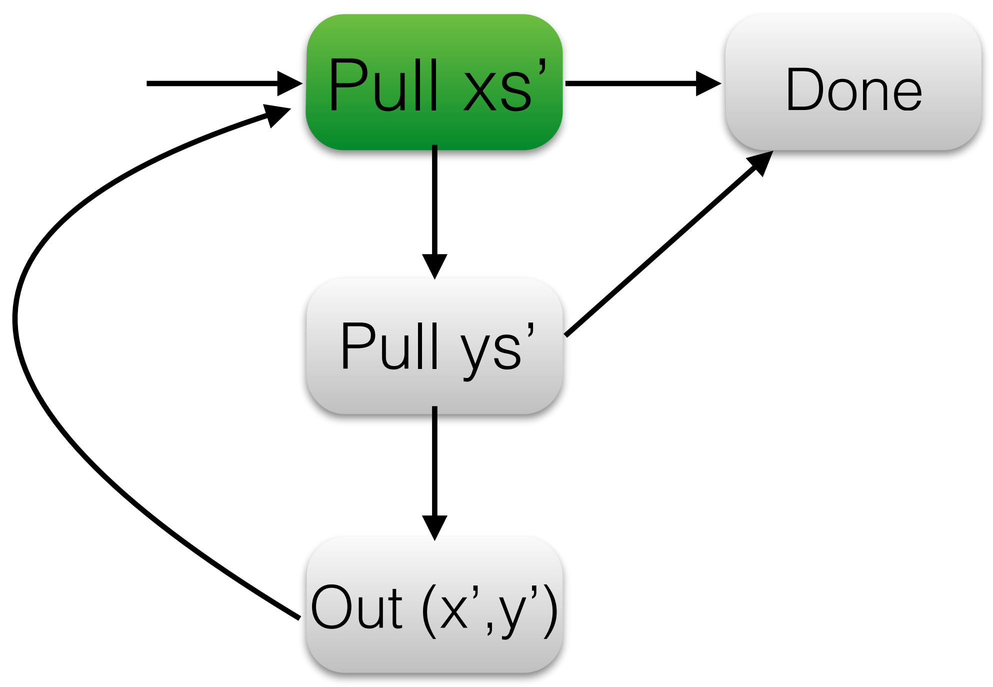



Pull xs
-
Done
If x > 0
Out x
Pull ys’
-
Pull xs
Out (x’,y’)
If x > 0
-
Out x

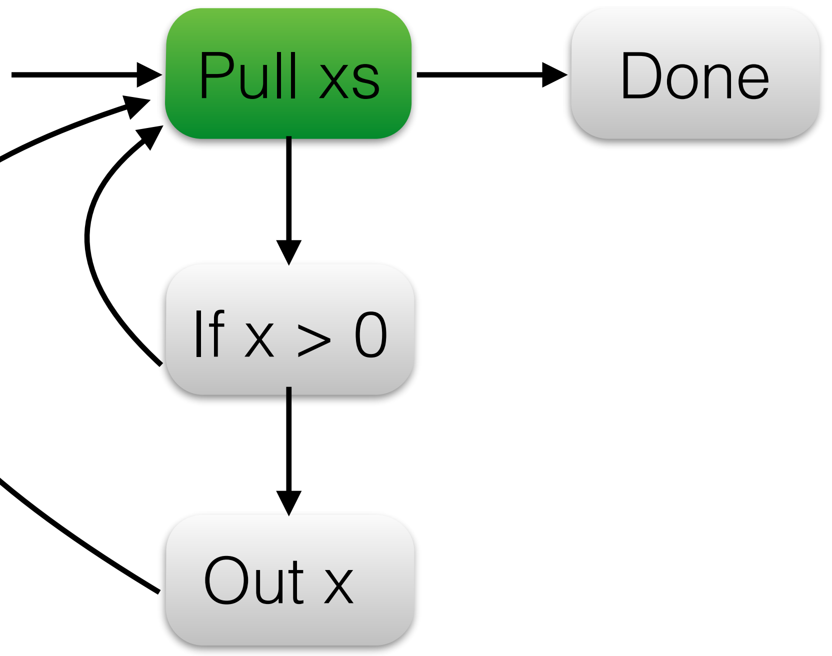
filter (>0) xs



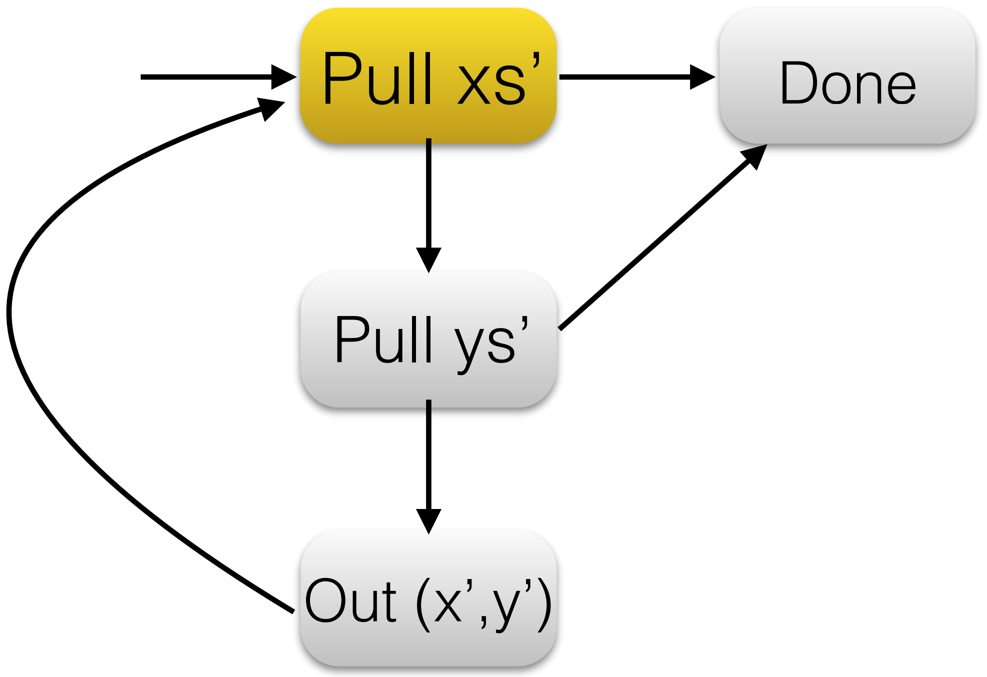
zip xs' ys'

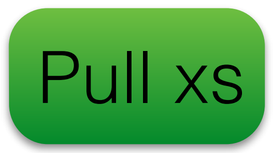


filter (>0) xs

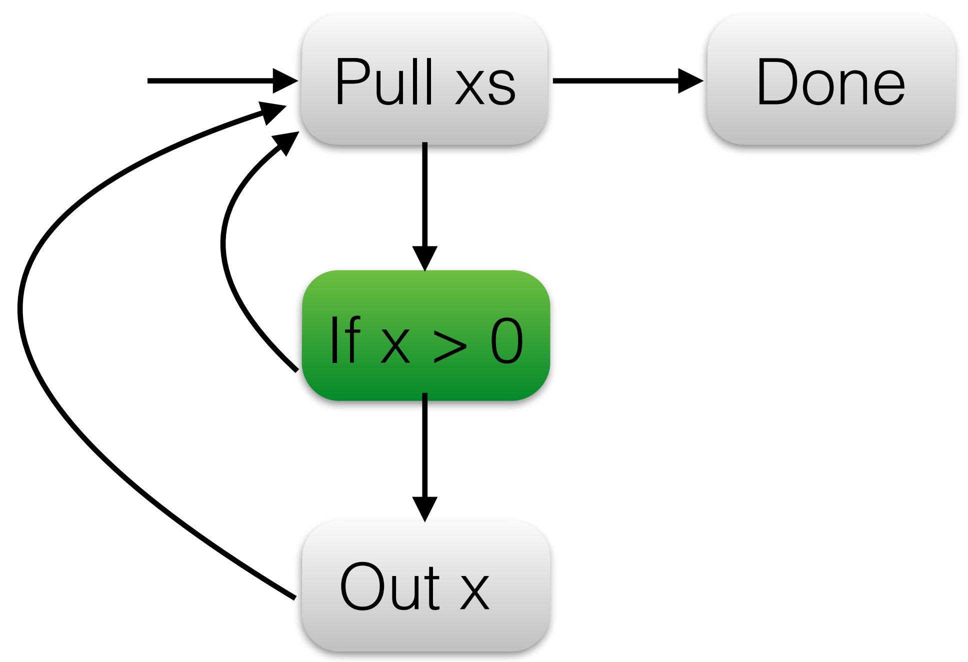


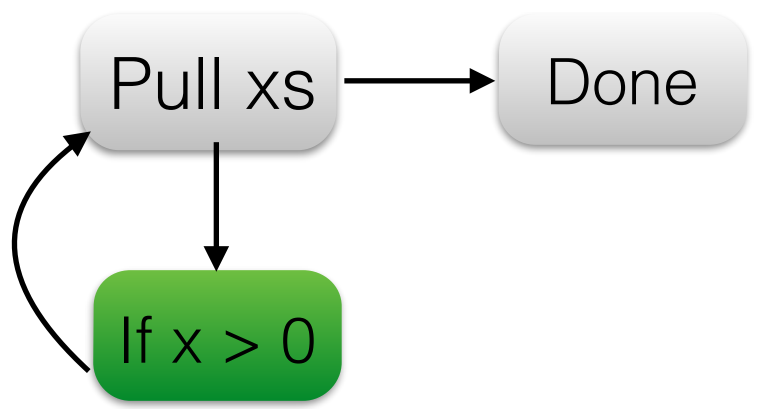
zip xs' ys'



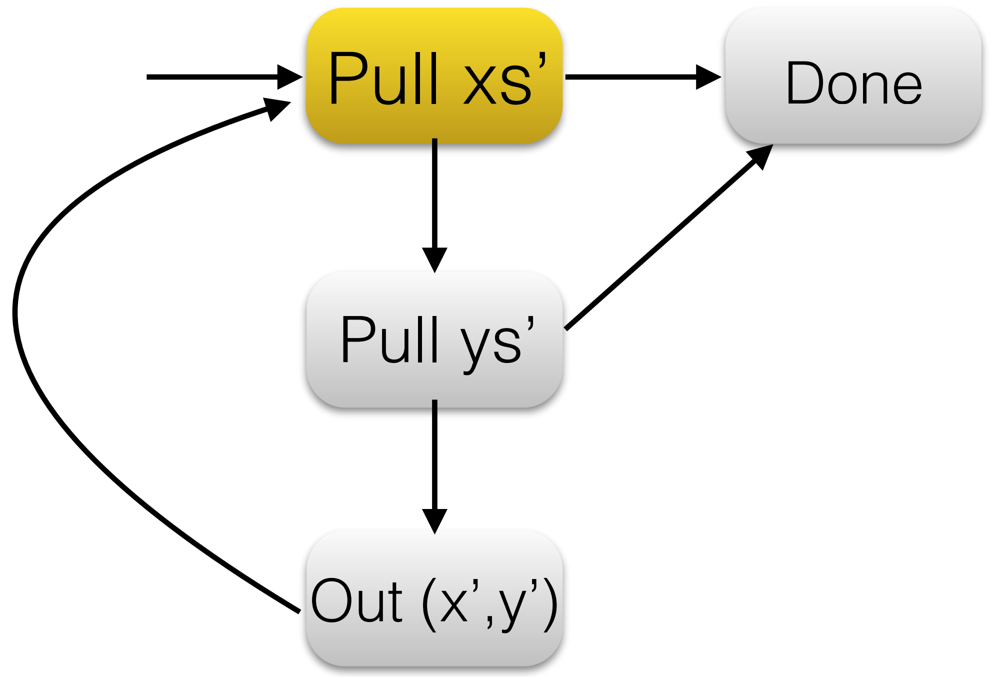


filter (>0) xs

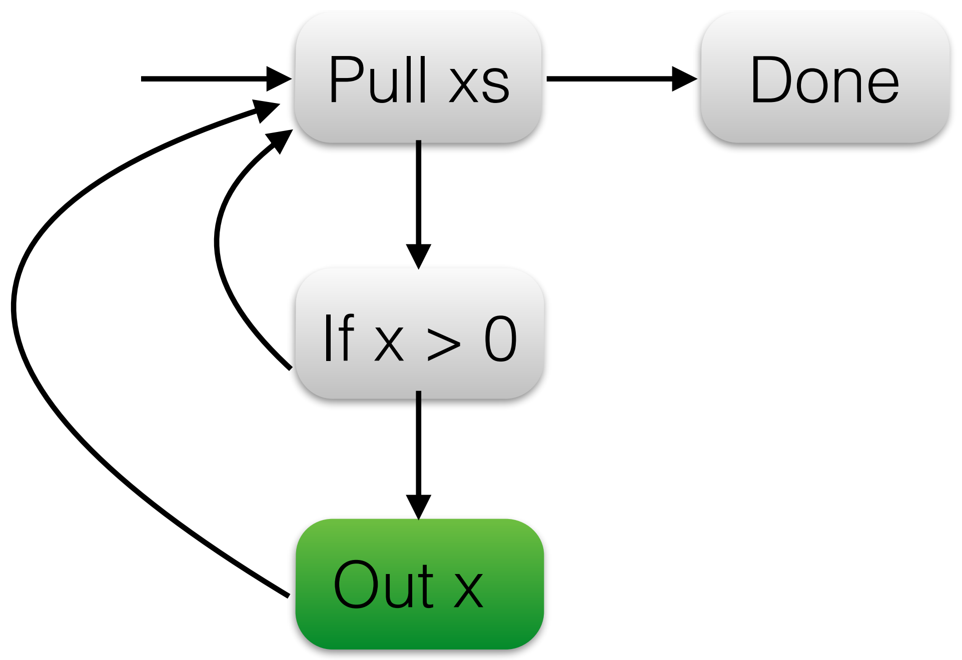


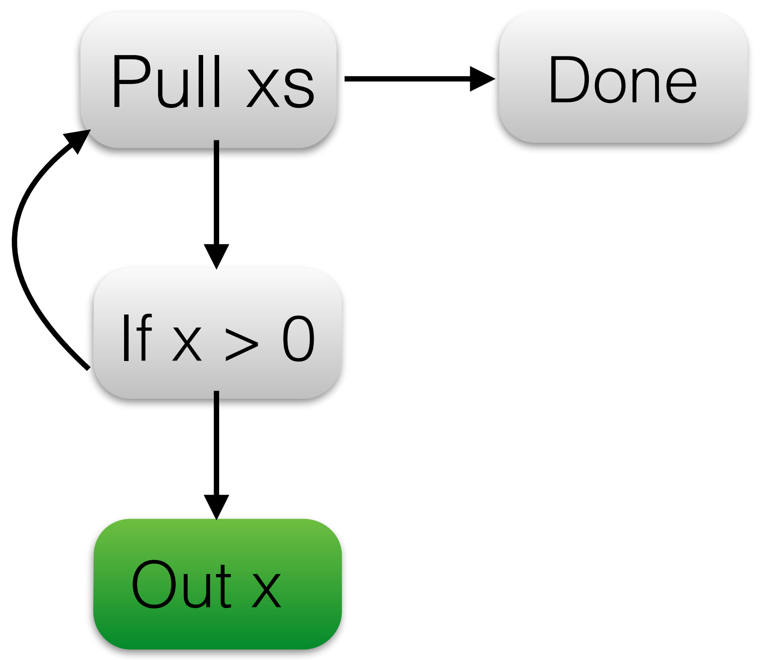


zip xs’ ys’

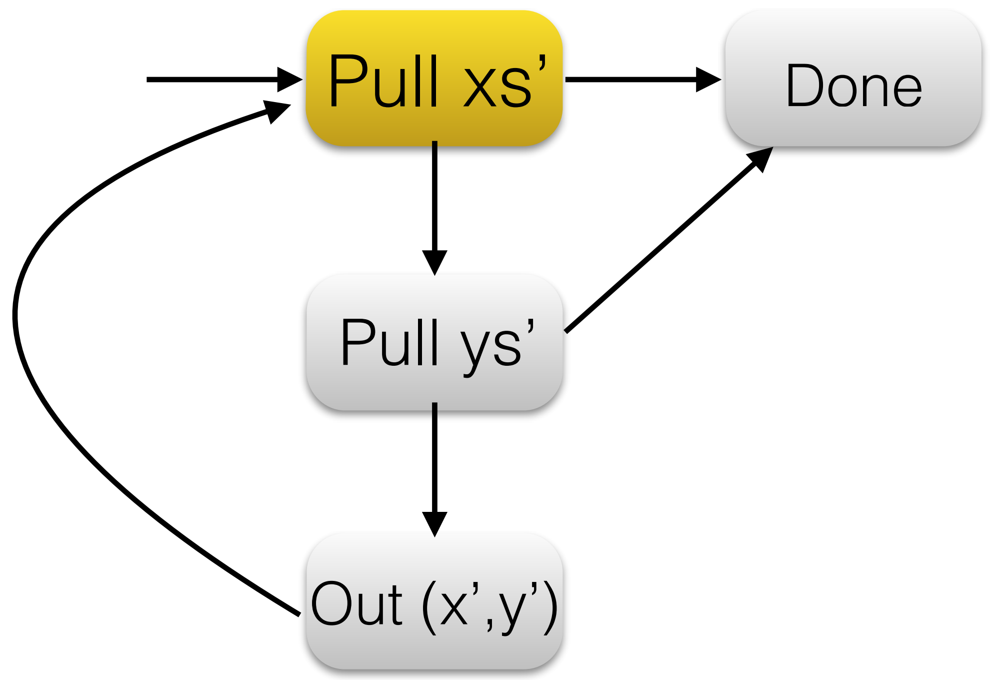


filter (>0) xs

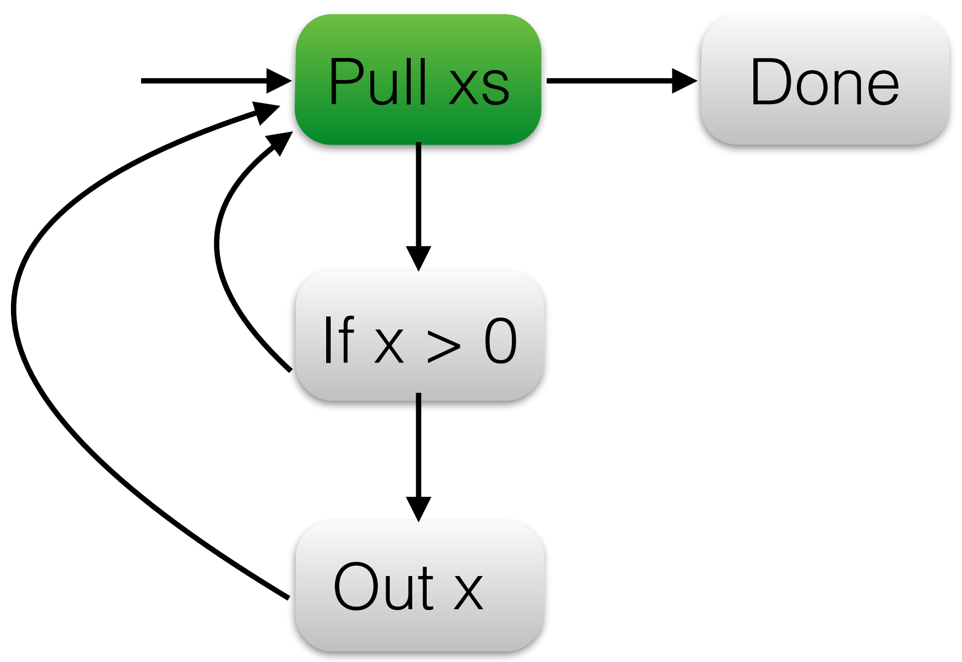


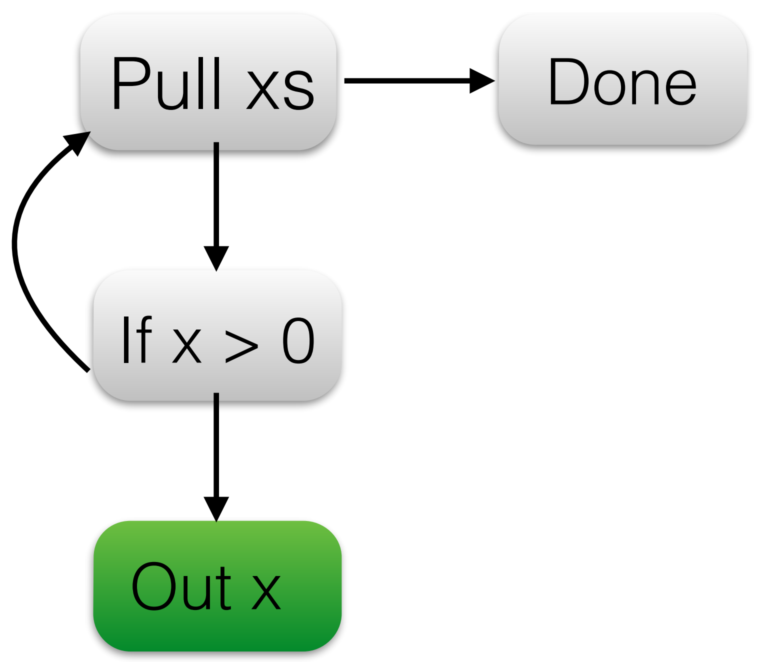


zip xs’ ys’

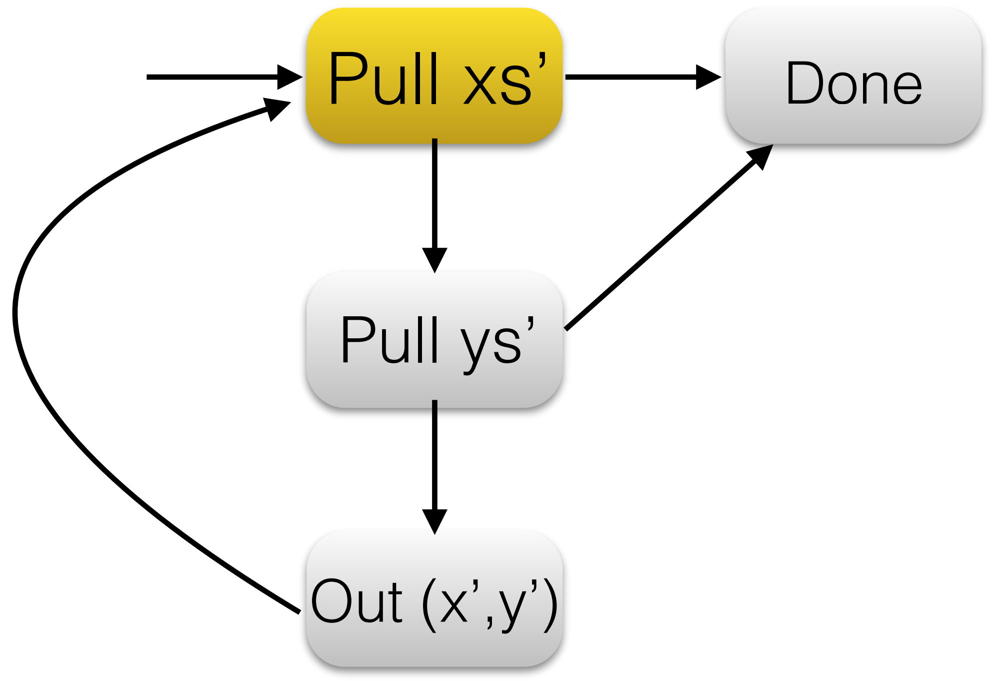


filter (>0) xs

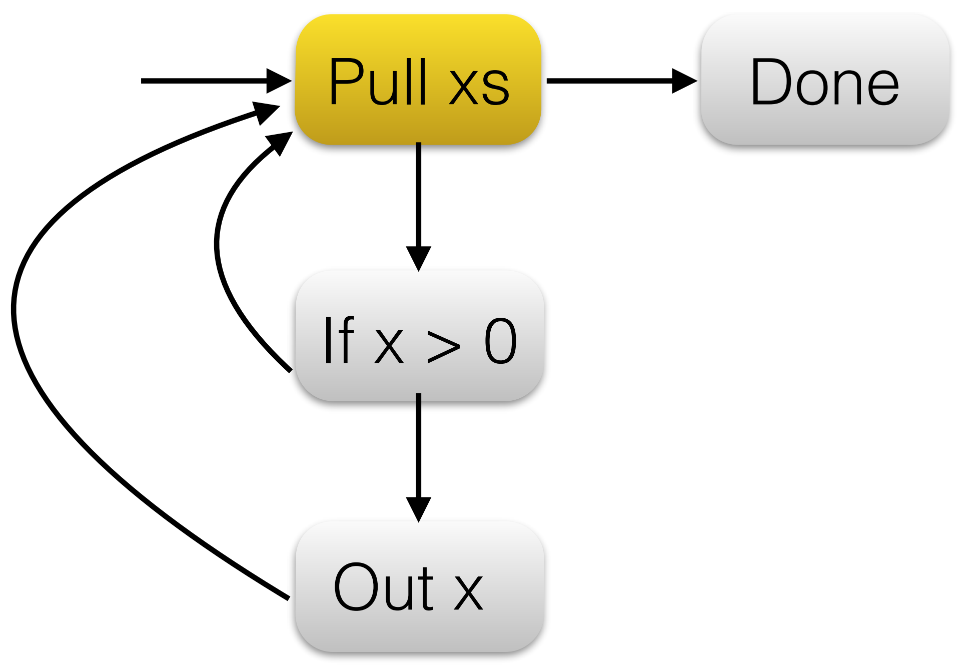


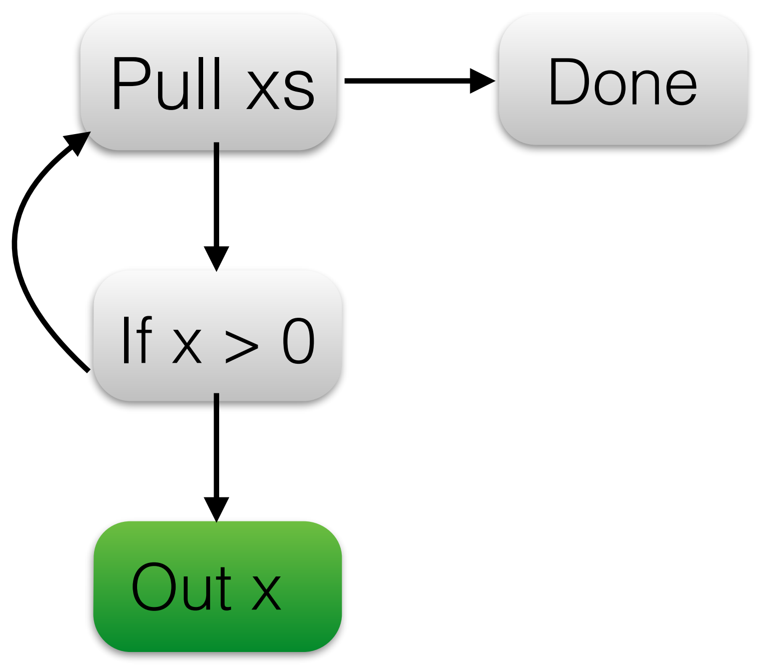


zip xs’ ys’

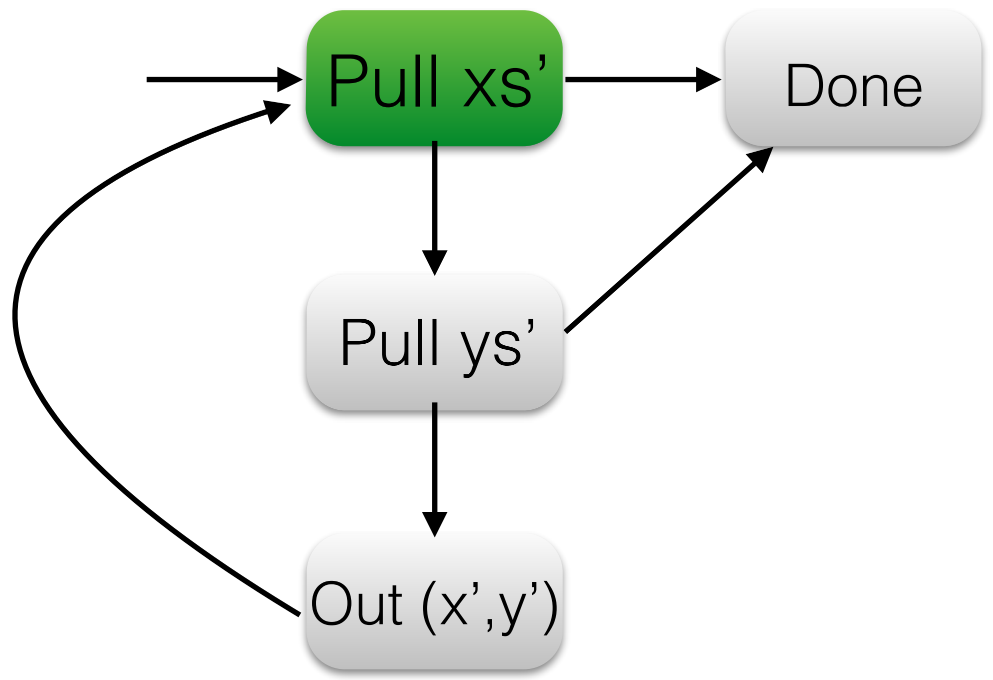


filter (>0) xs

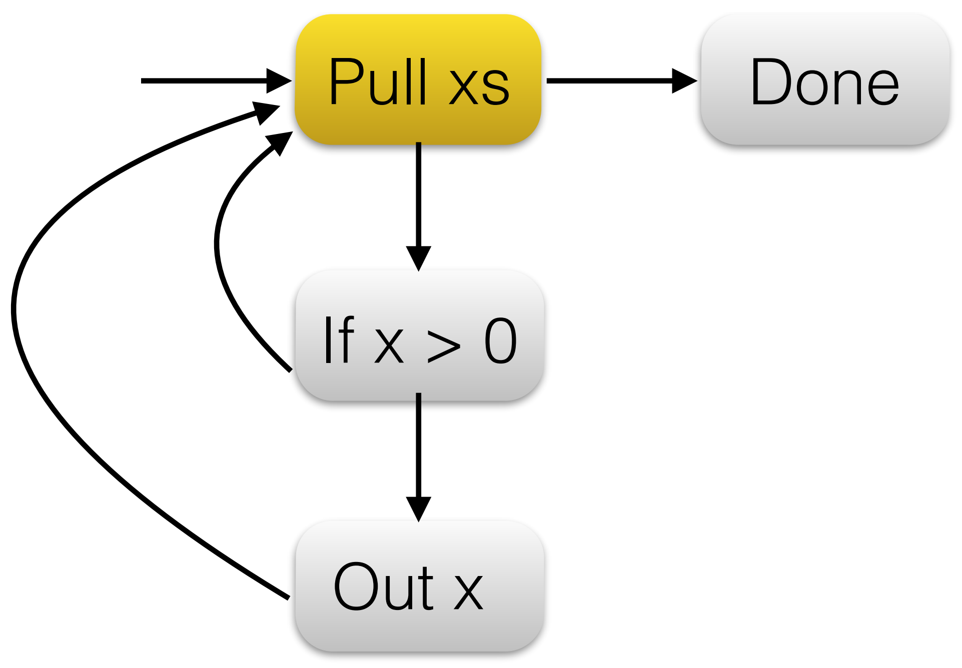




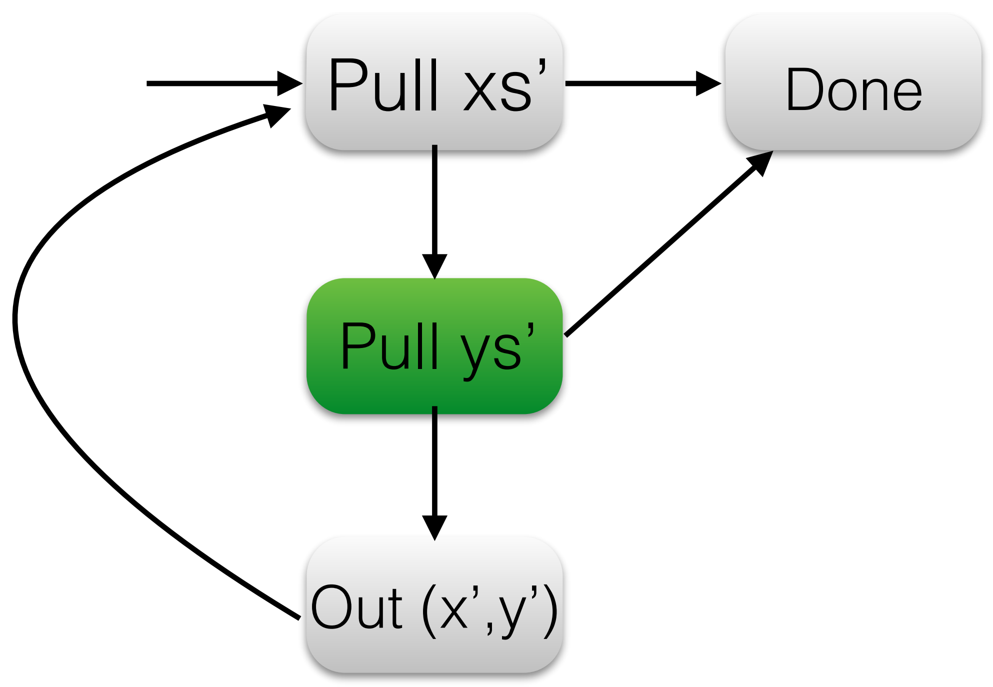
zip xs’ ys’

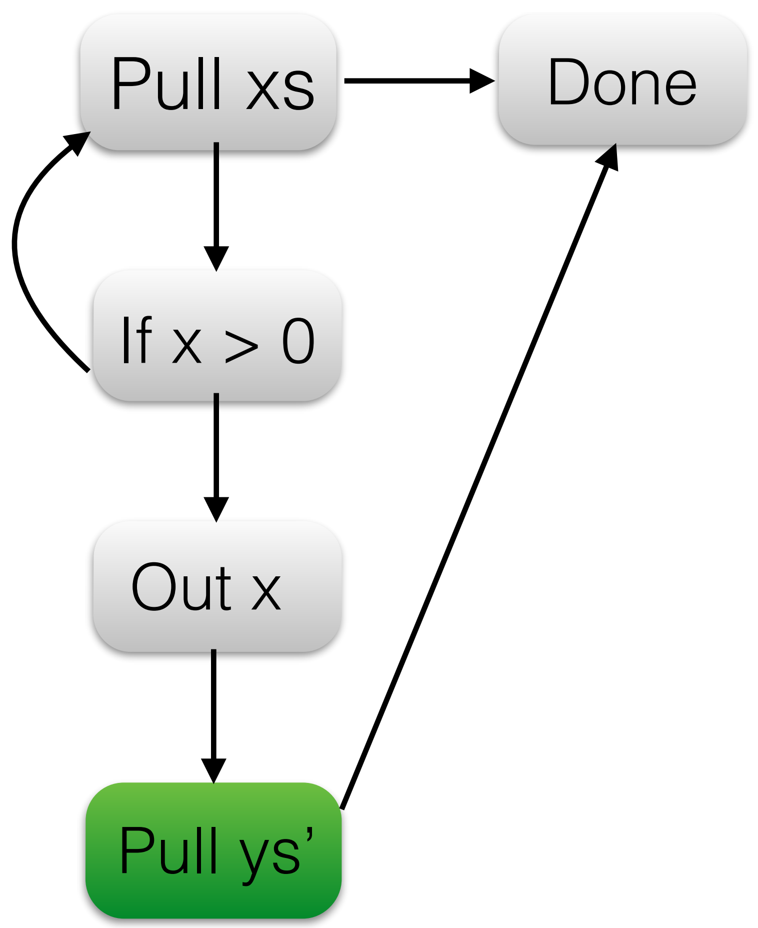


filter (>0) xs

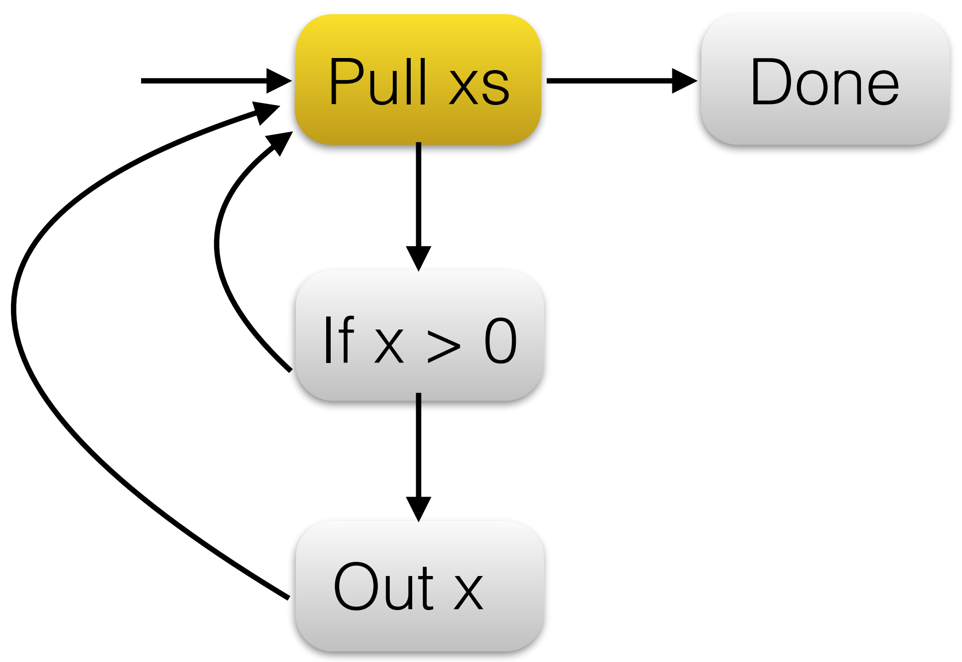


zip xs’ ys’

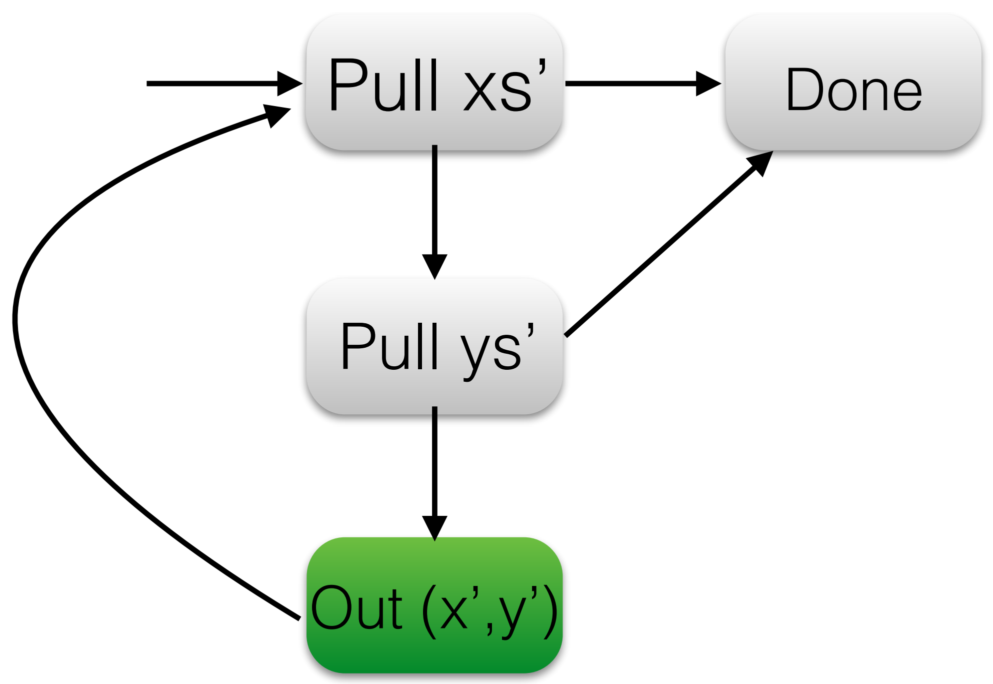


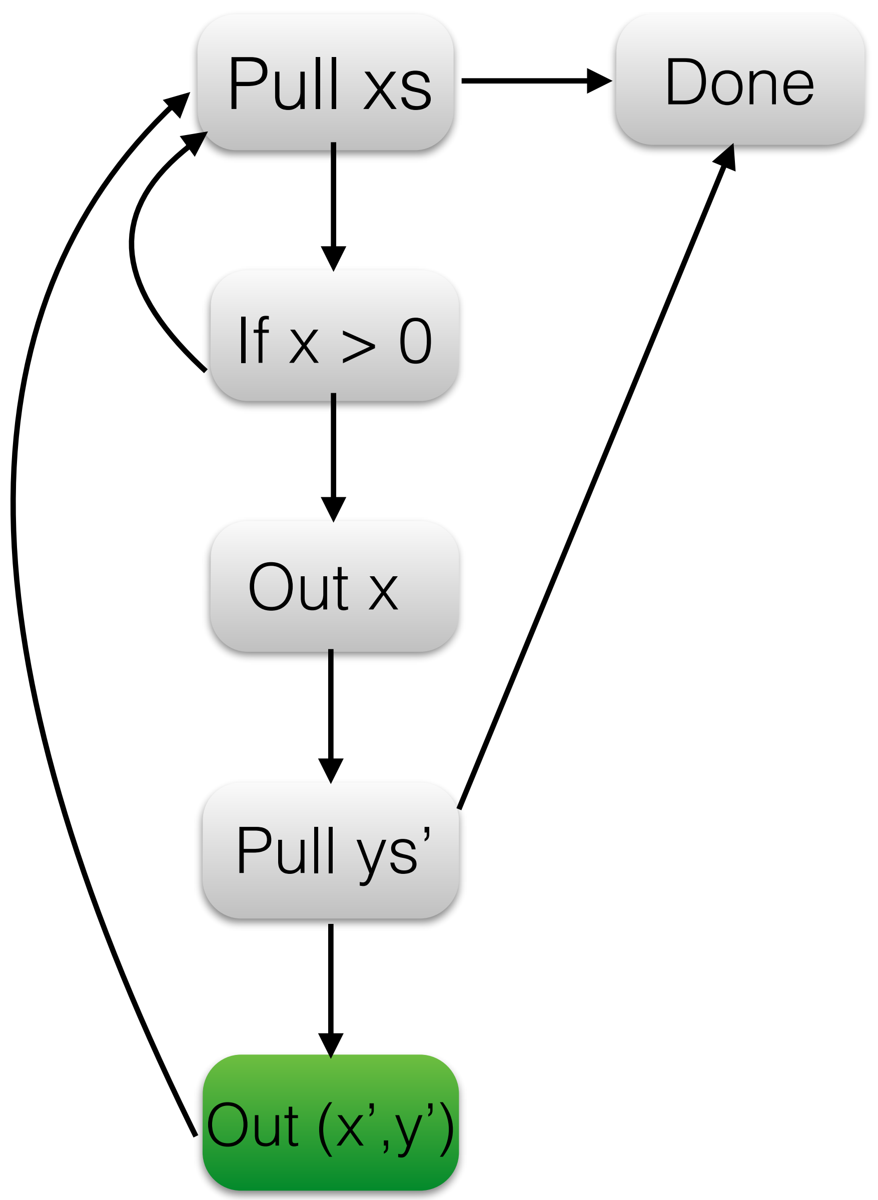


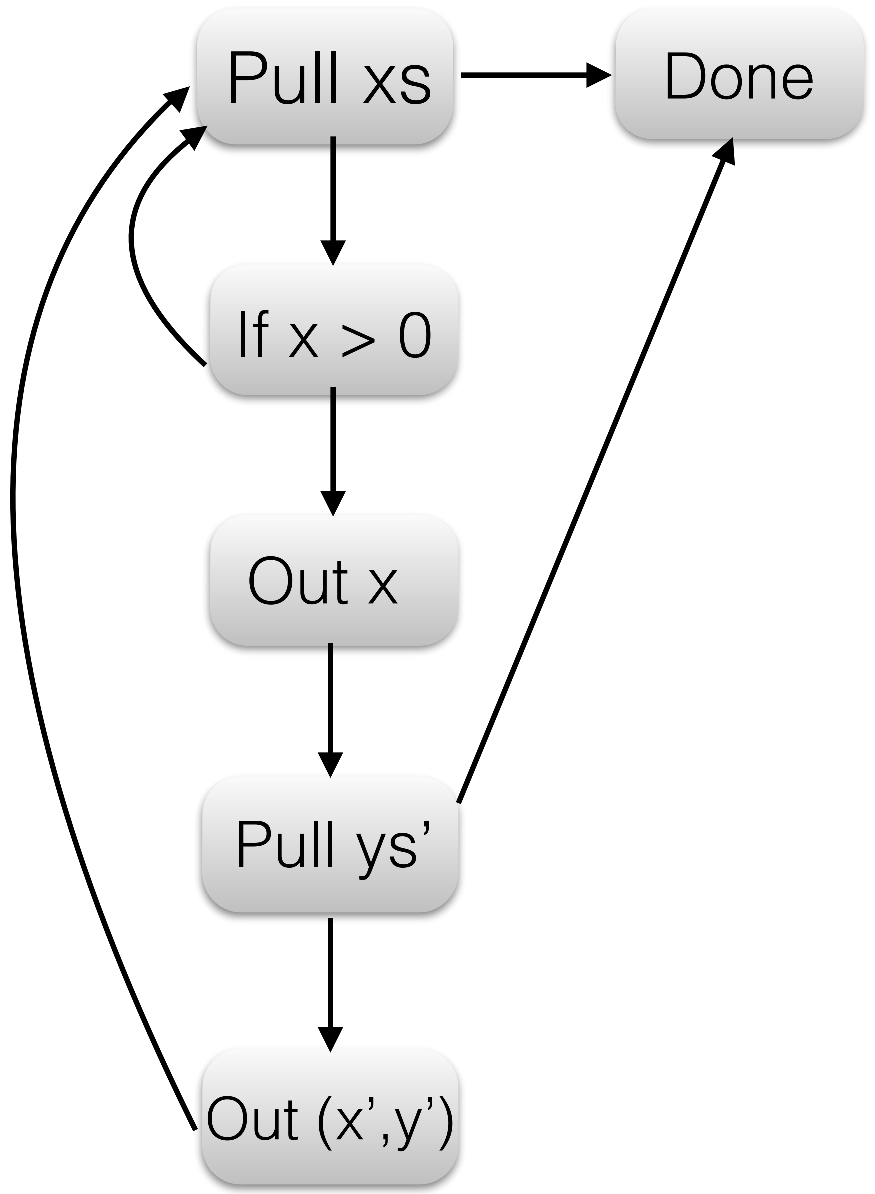
filter (>0) xs



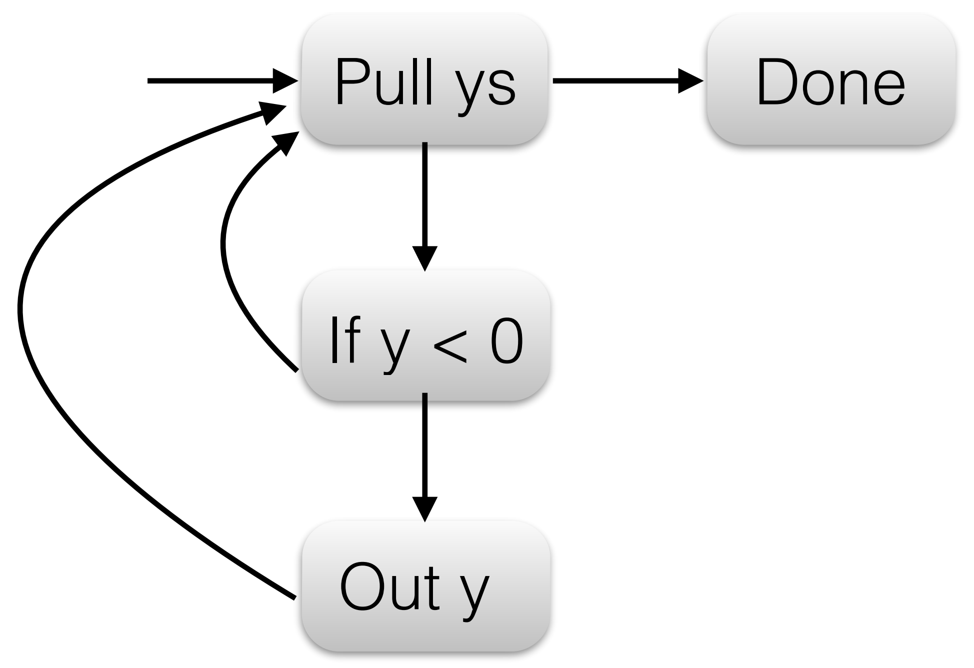
zip xs’ ys’

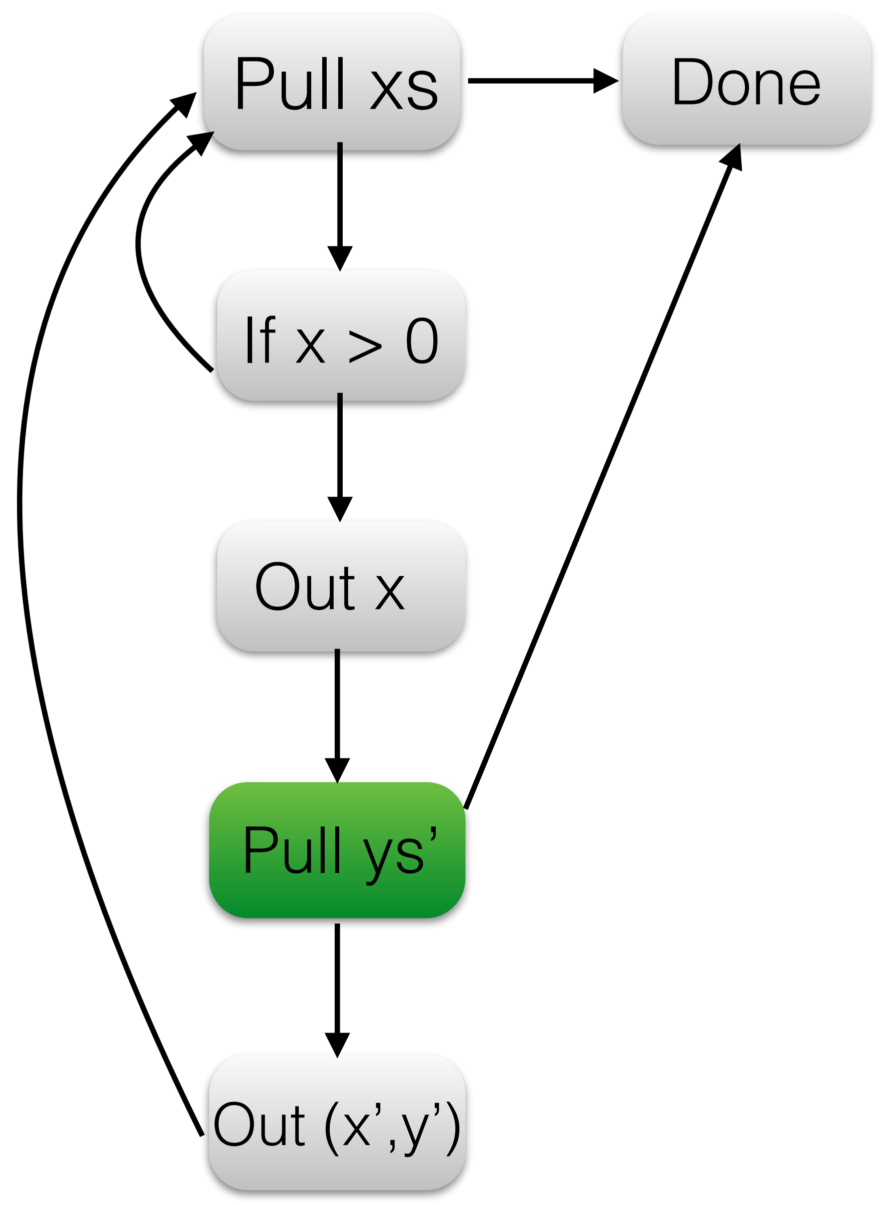




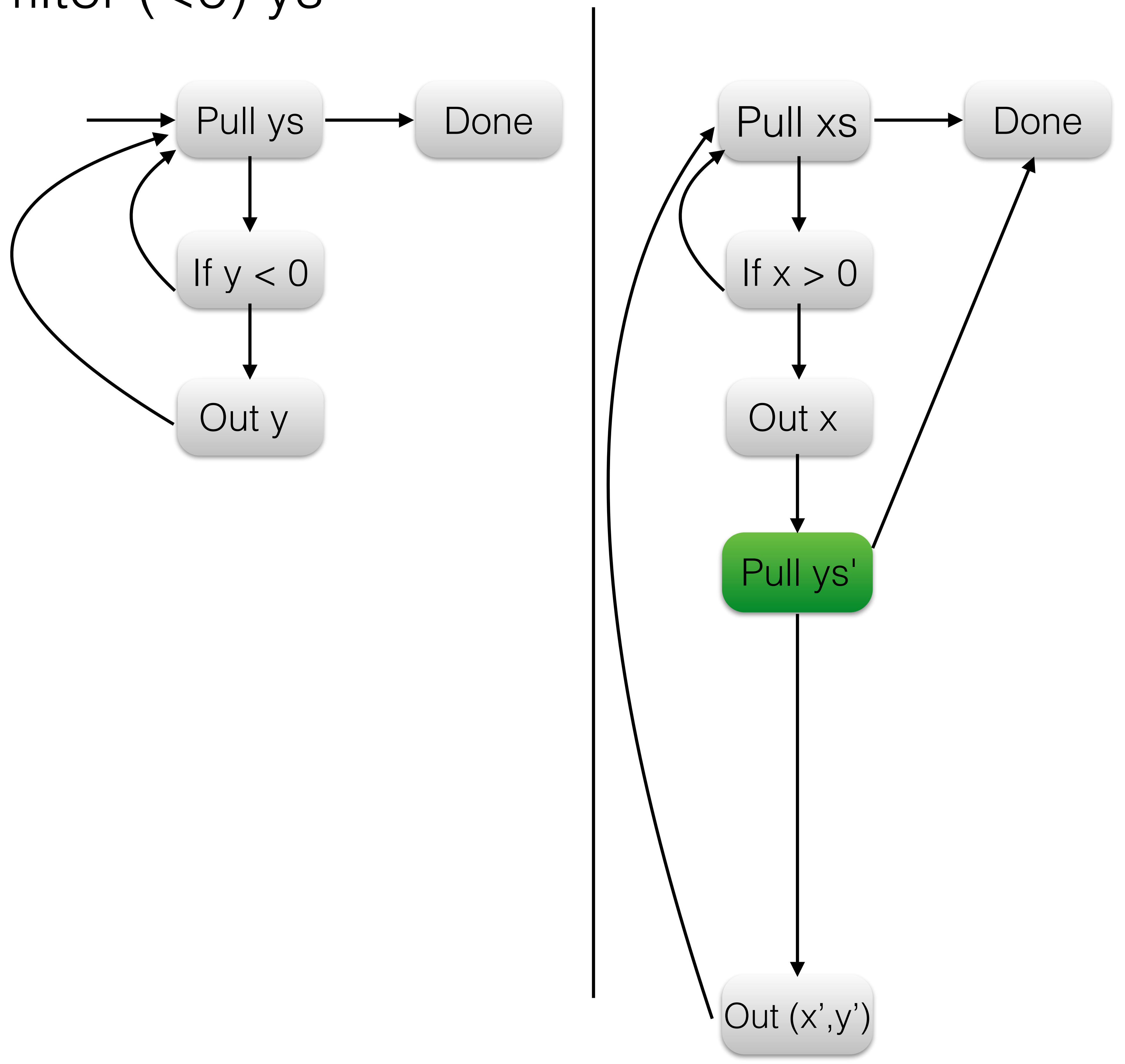

Pull xs
Done
If x > 0
Out x
Pull ys’
Out (x’,y’)

filter (<0) ys

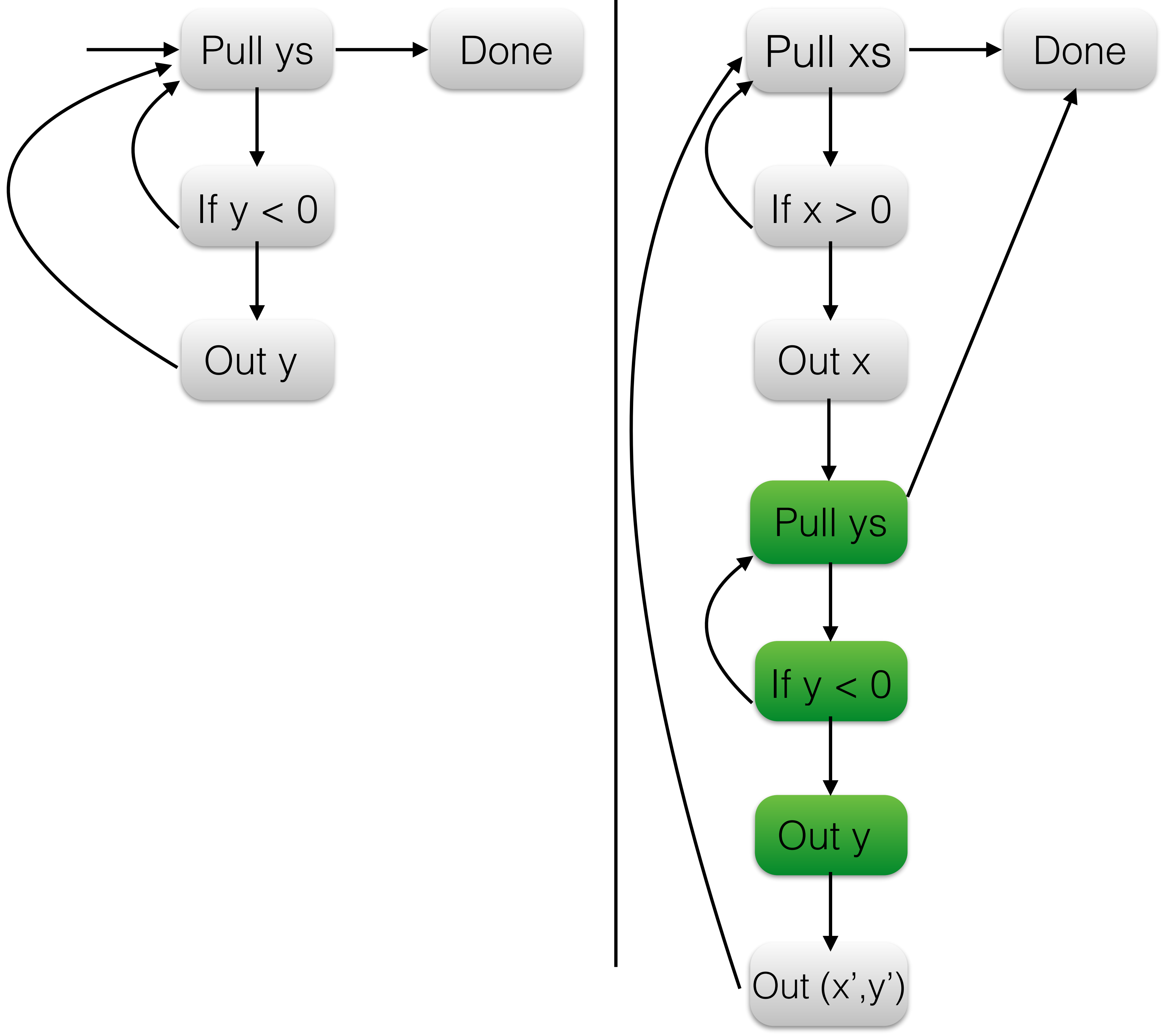


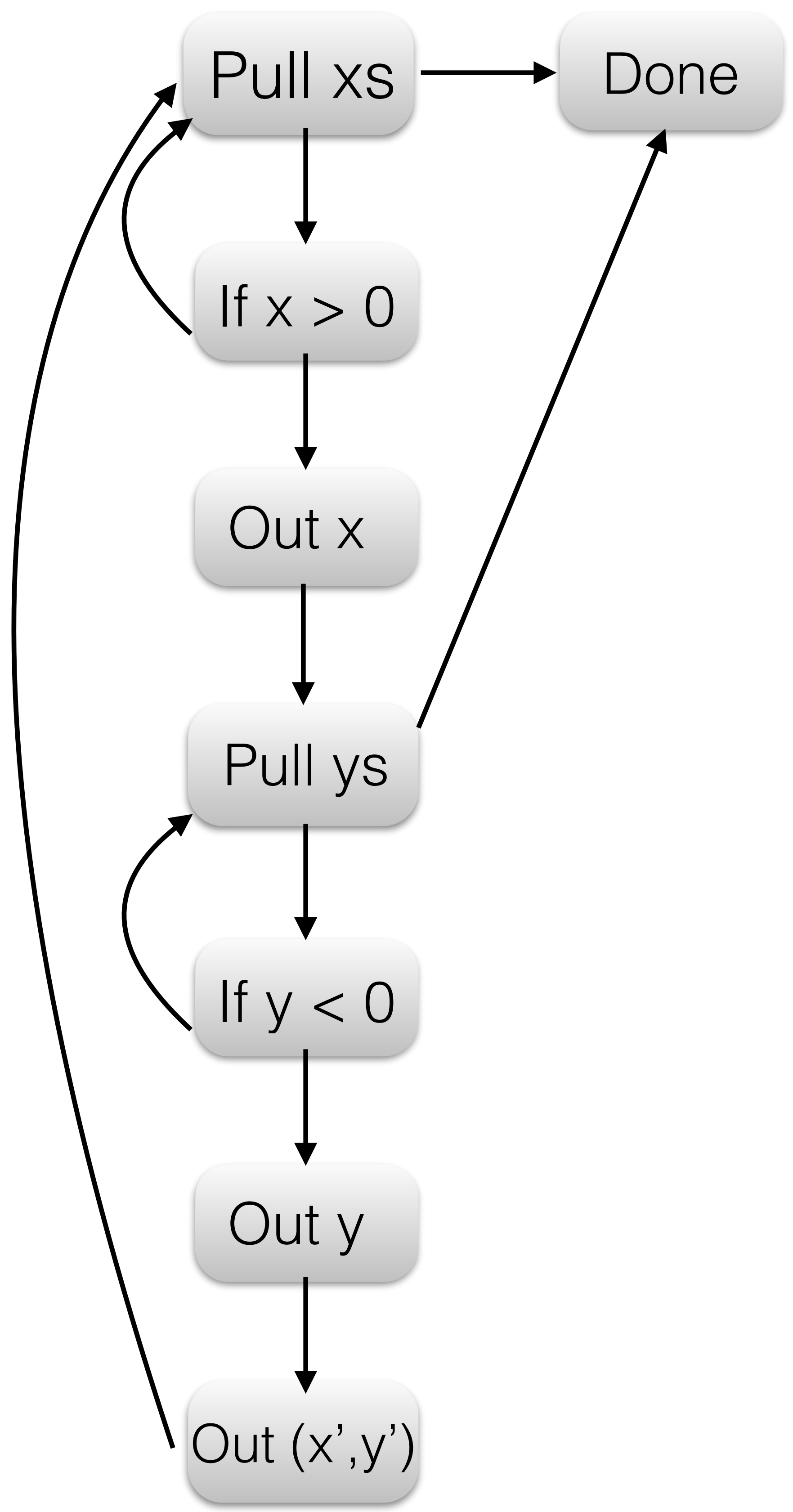


filter (<0) ys

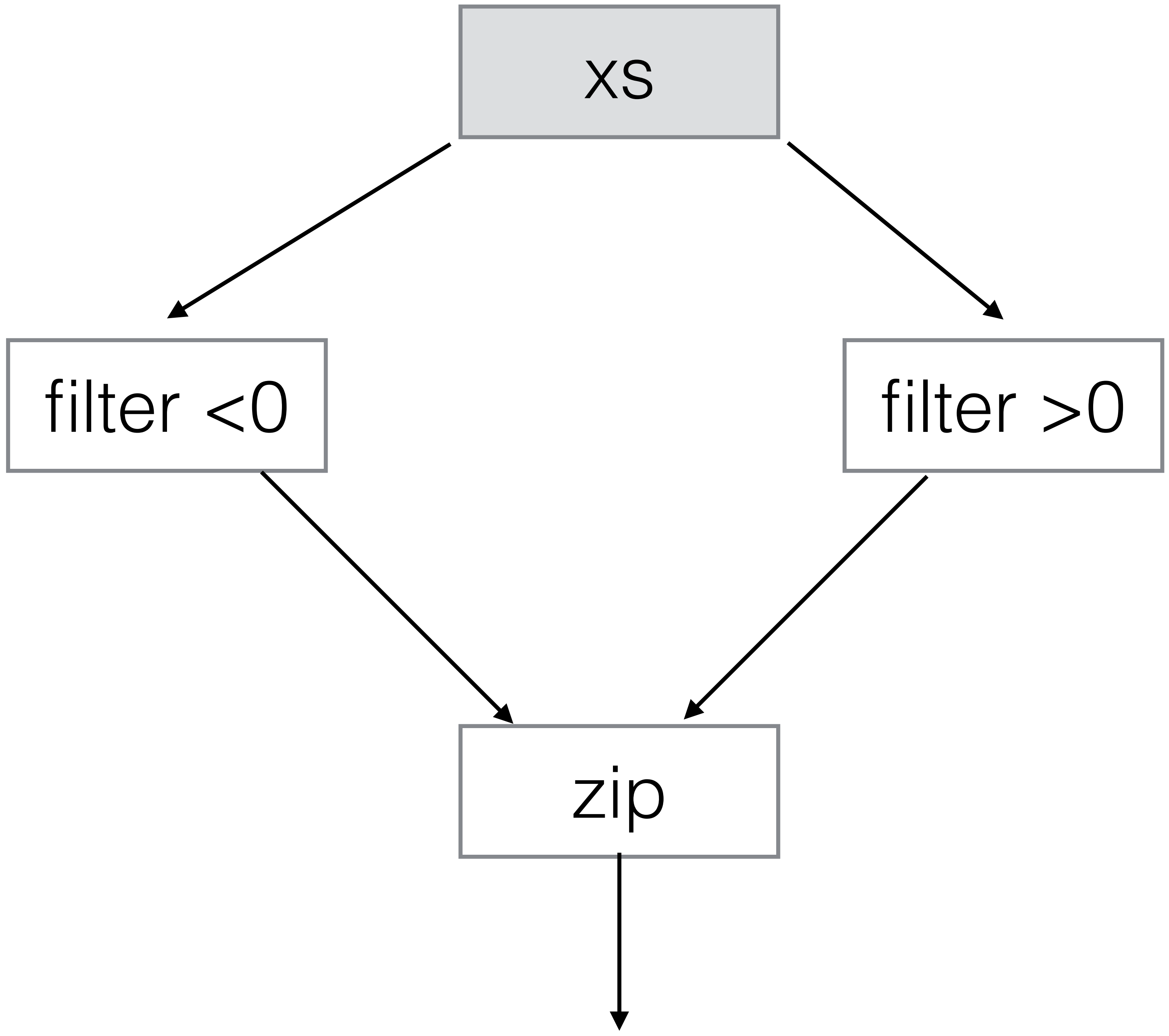


filter (<0) ys

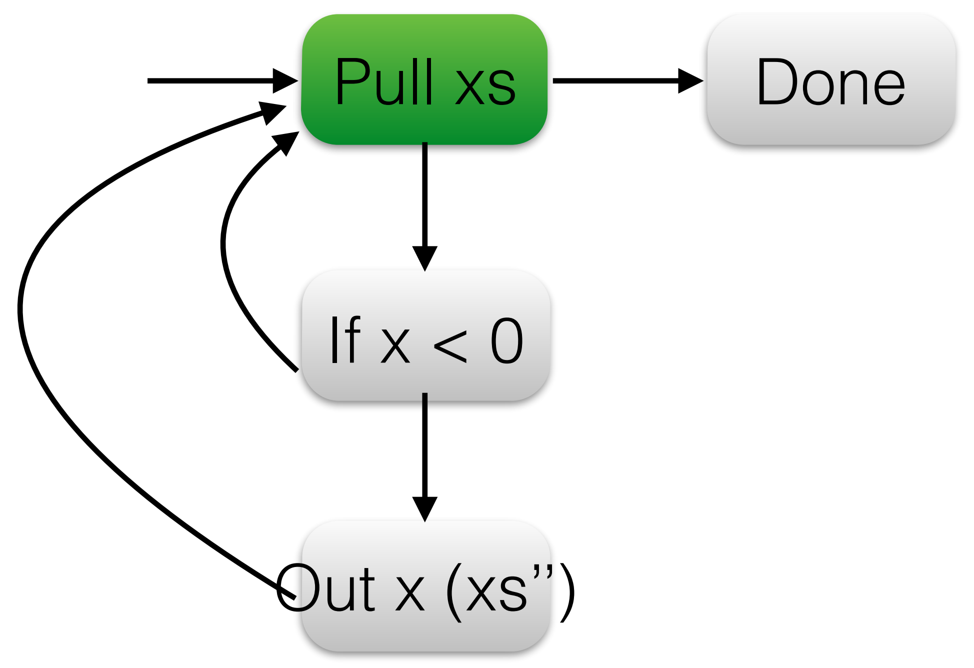

Pull ys
Done
If y < 0
Out y
Pull xs
Done
If x > 0
Out x
Pull ys
If y < 0
Out y
Out (x',y')


Pull xs
Done
If x > 0
Out x
Pull ys
If y < 0
Out y
Out (x',y')

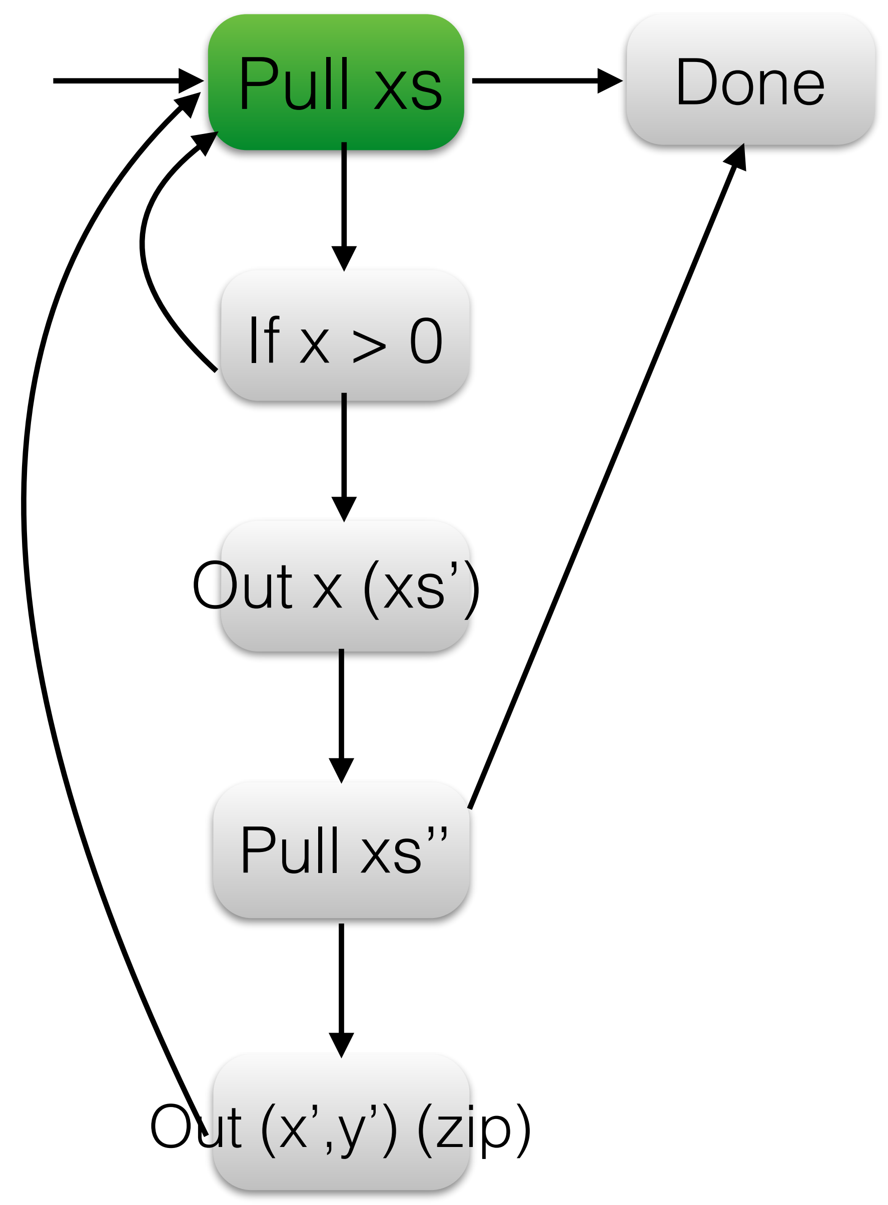
Bad example: requires buffer



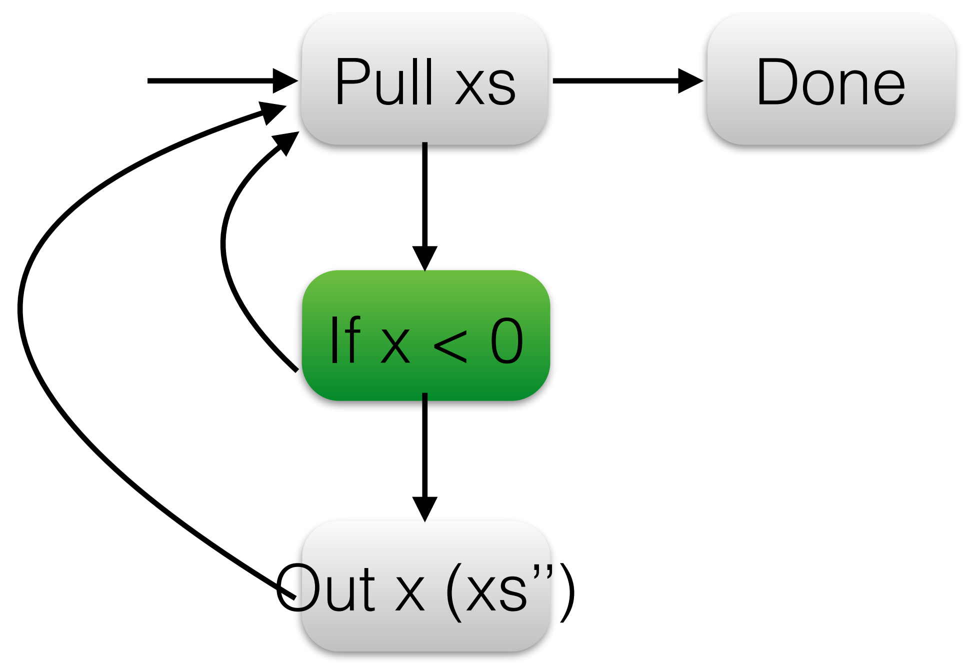
filter (<0) xs



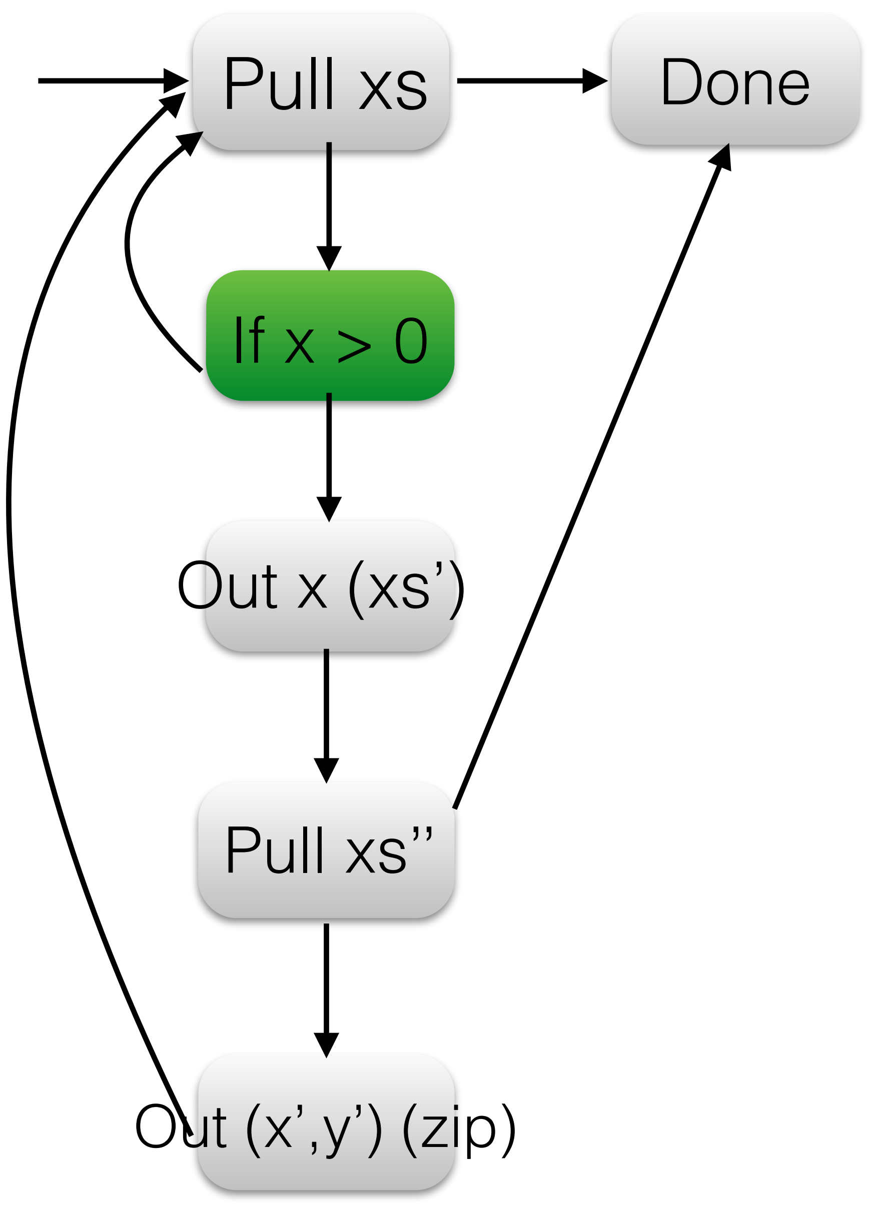
zip (filter (>0) xs) xs''



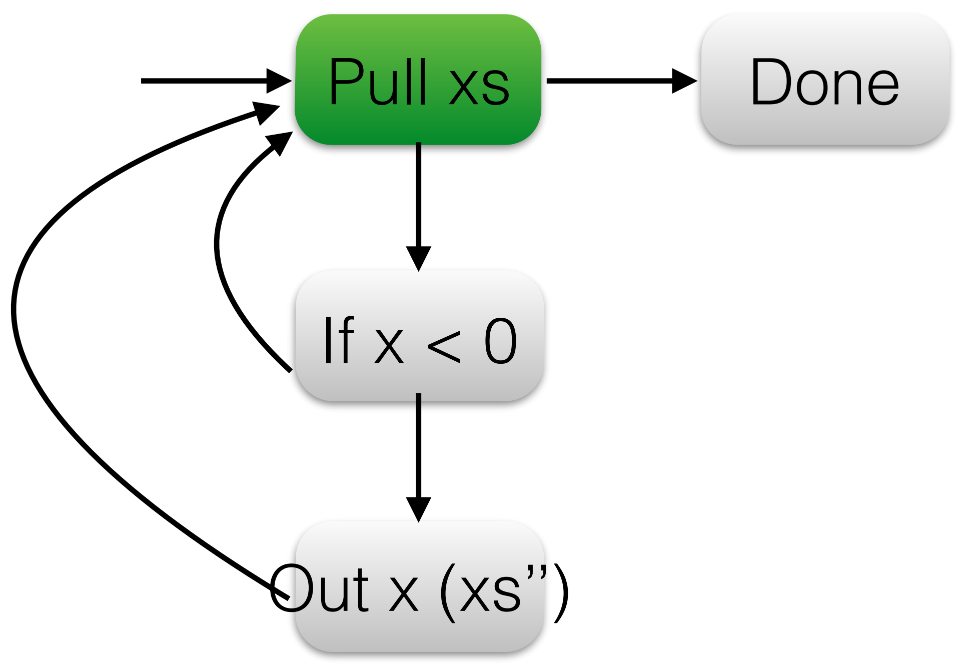
filter (<0) xs



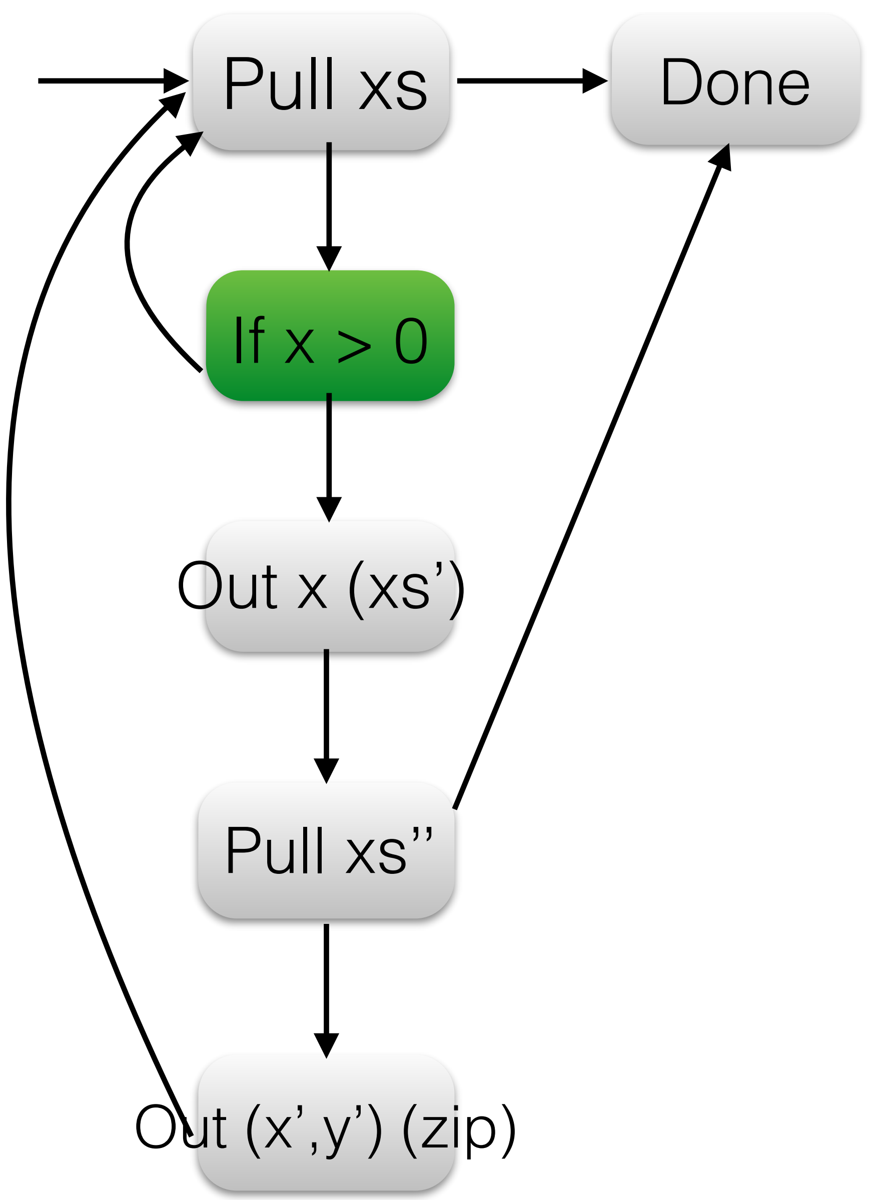
zip (filter (>0) xs) xs''



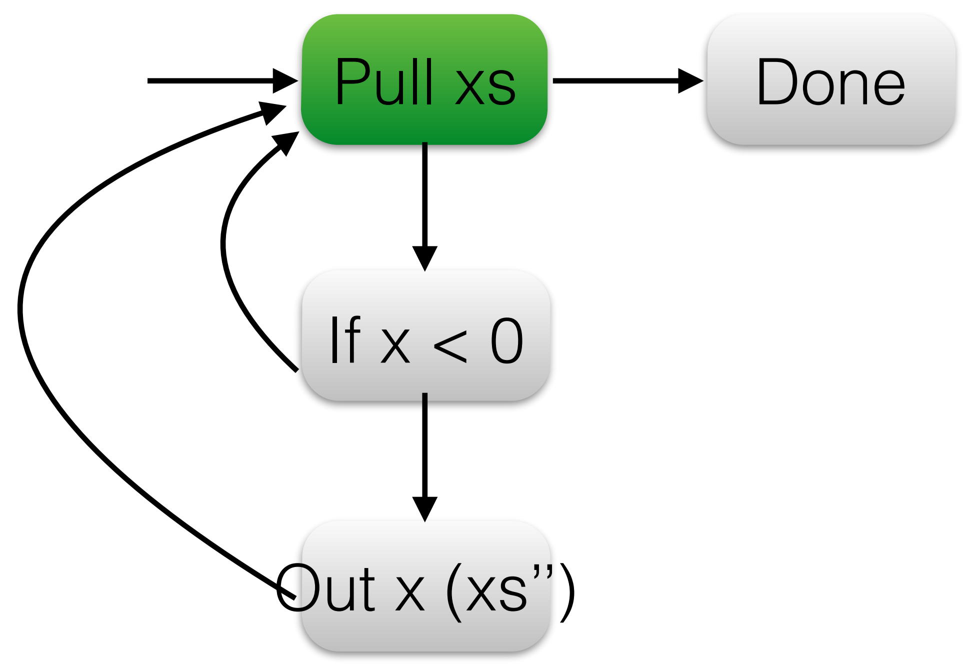
filter (<0) xs



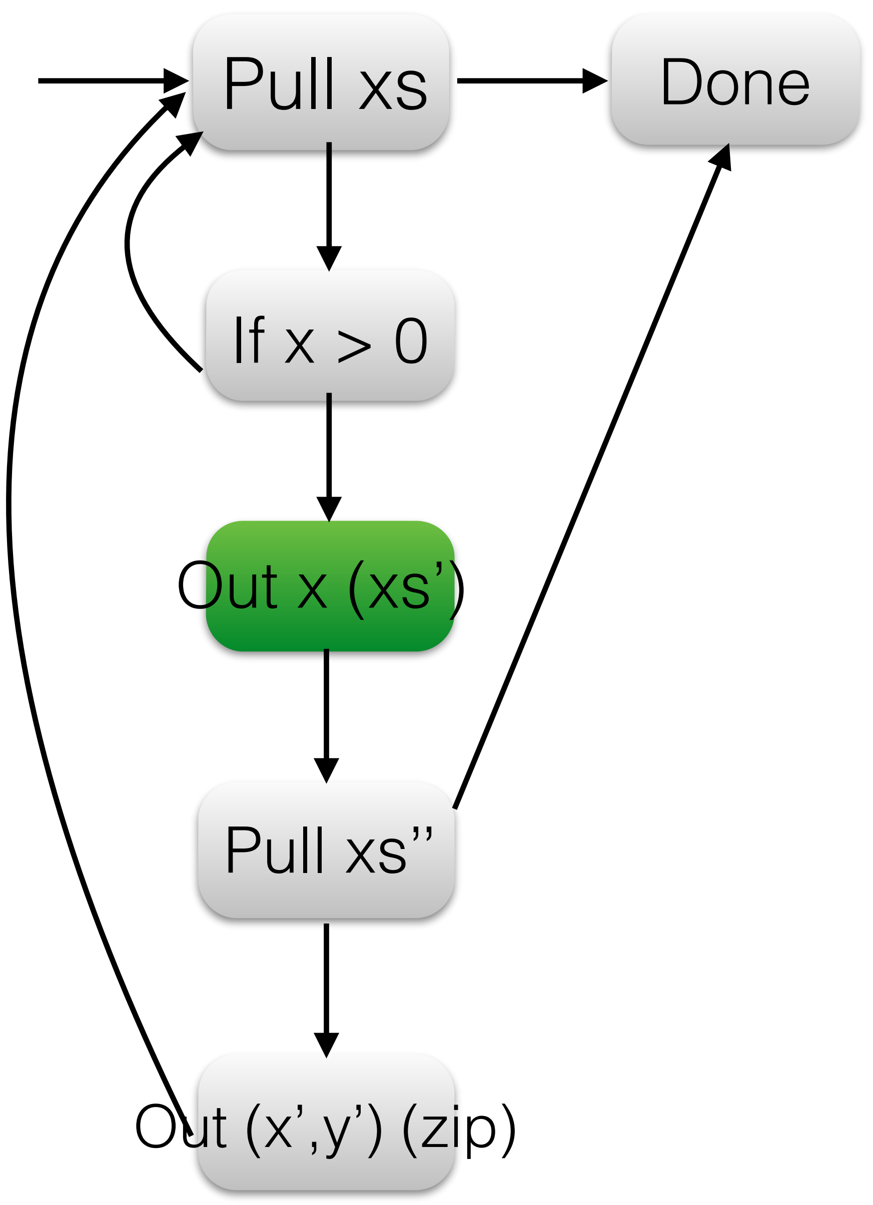
zip (filter (>0) xs) xs''



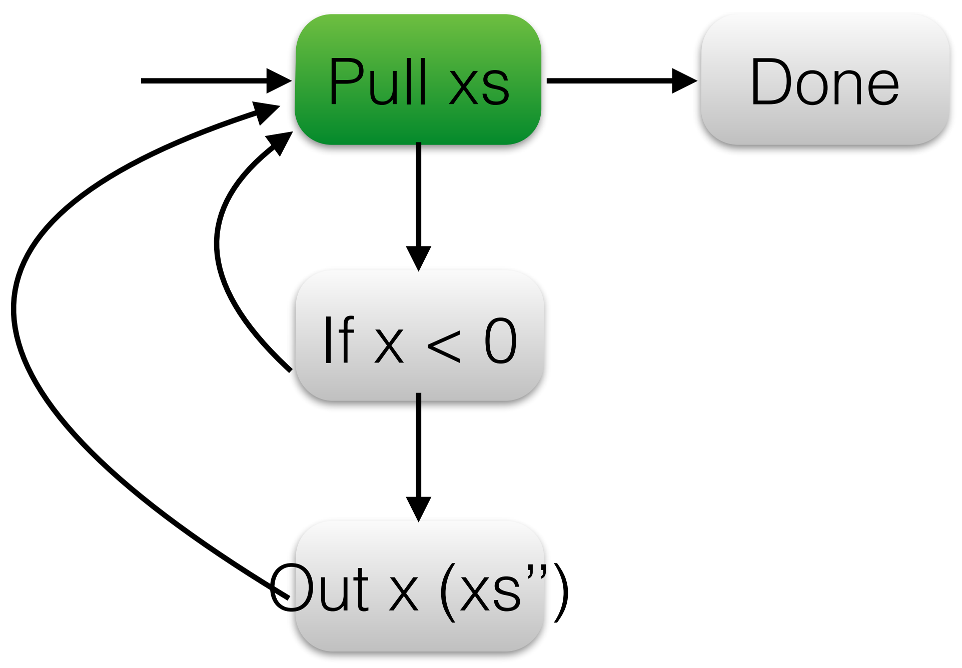
filter (<0) xs



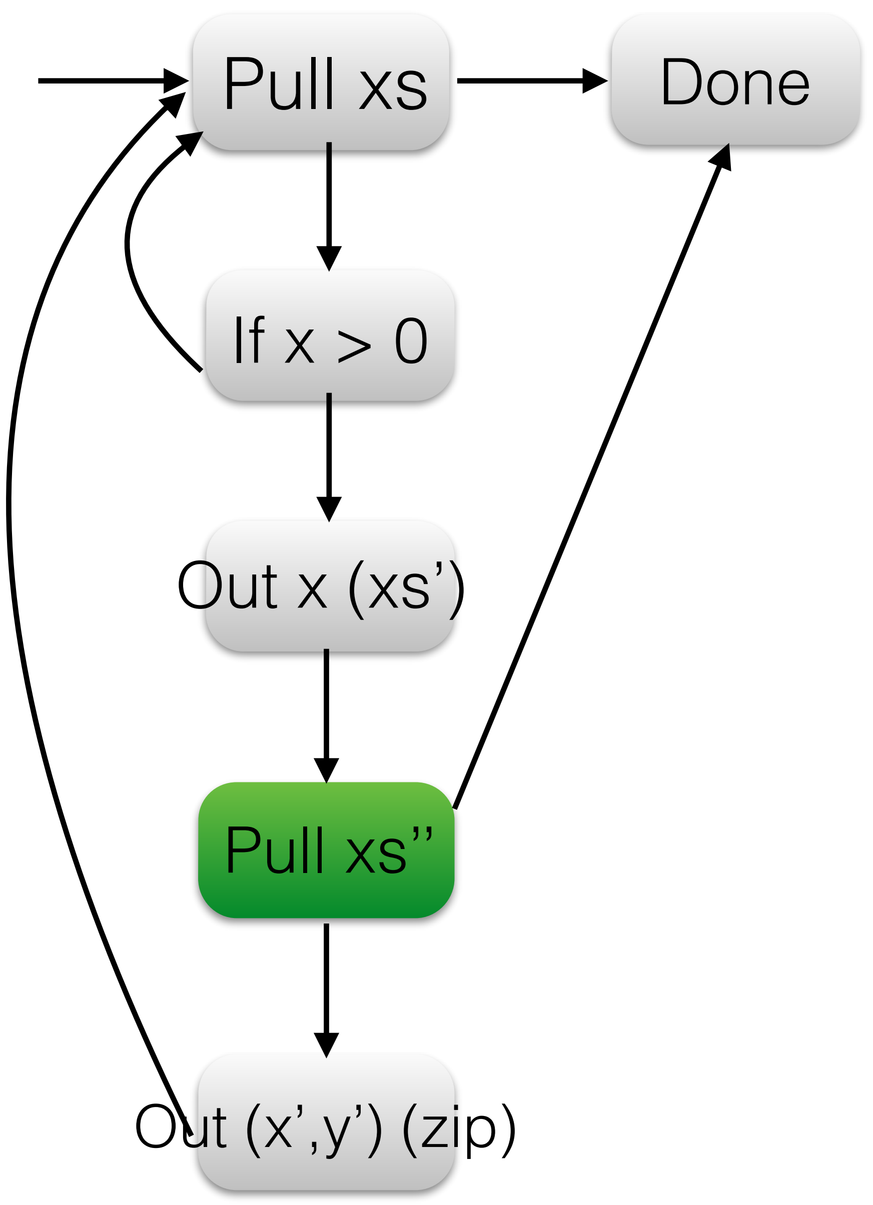
zip (filter (>0) xs) xs''



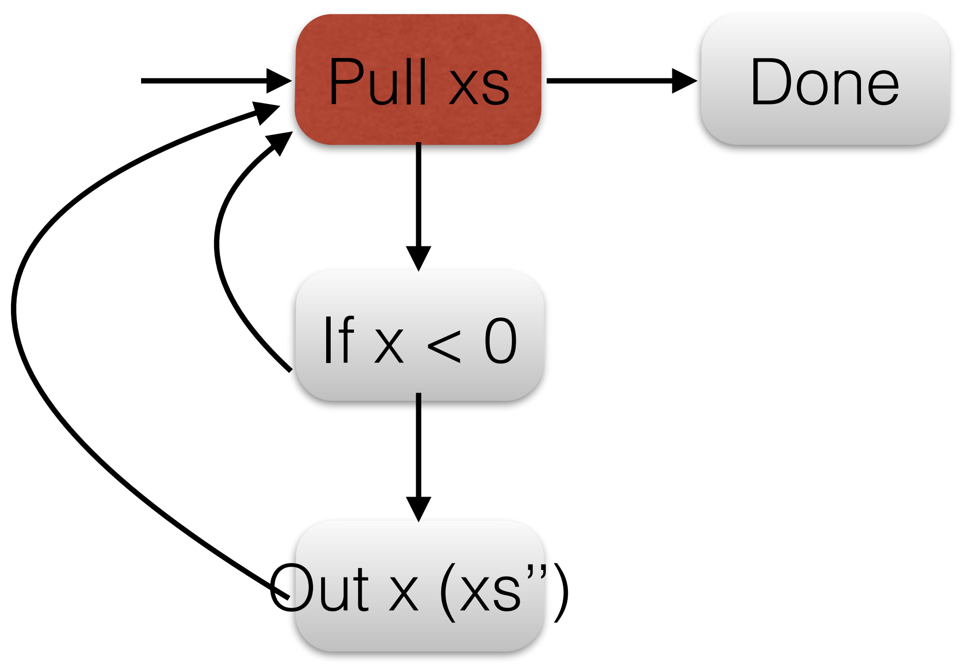
filter (<0) xs



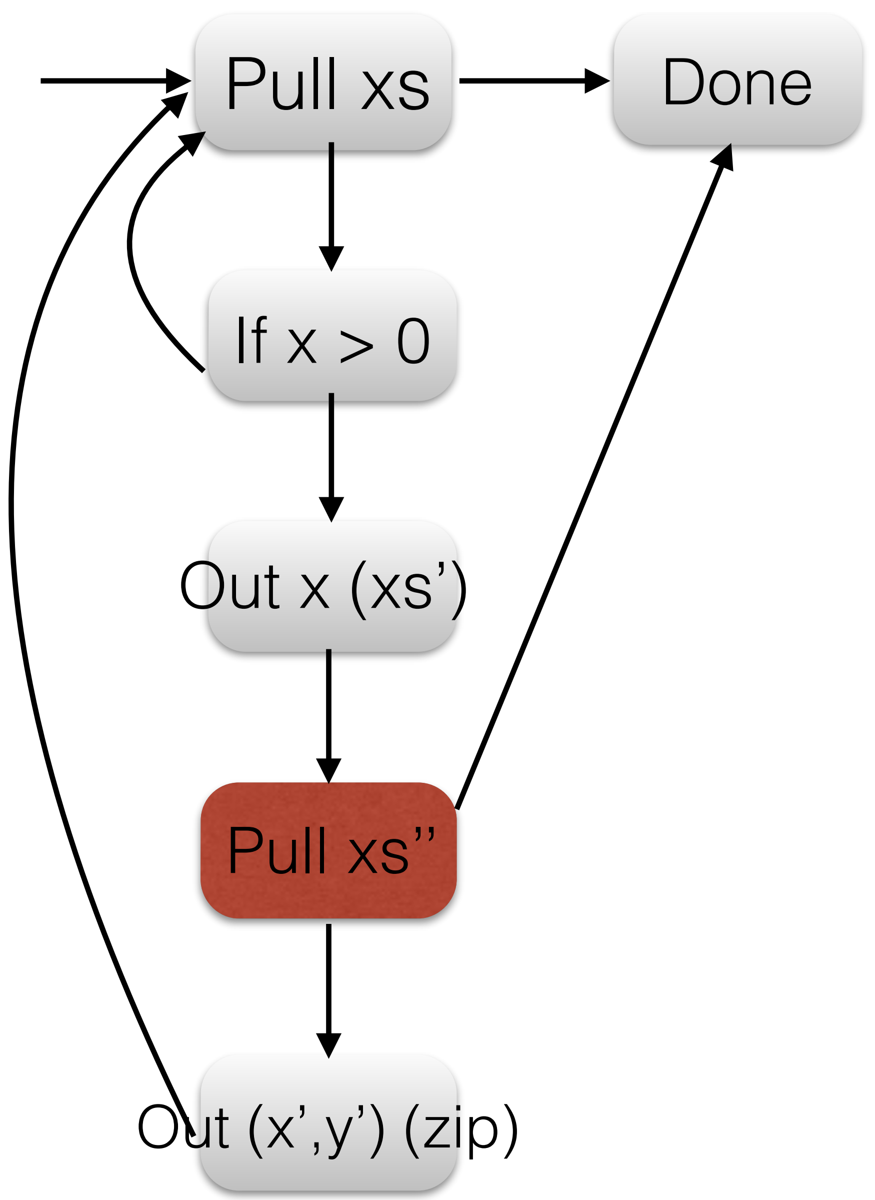
zip (filter (>0) xs) xs''



filter (<0) xs

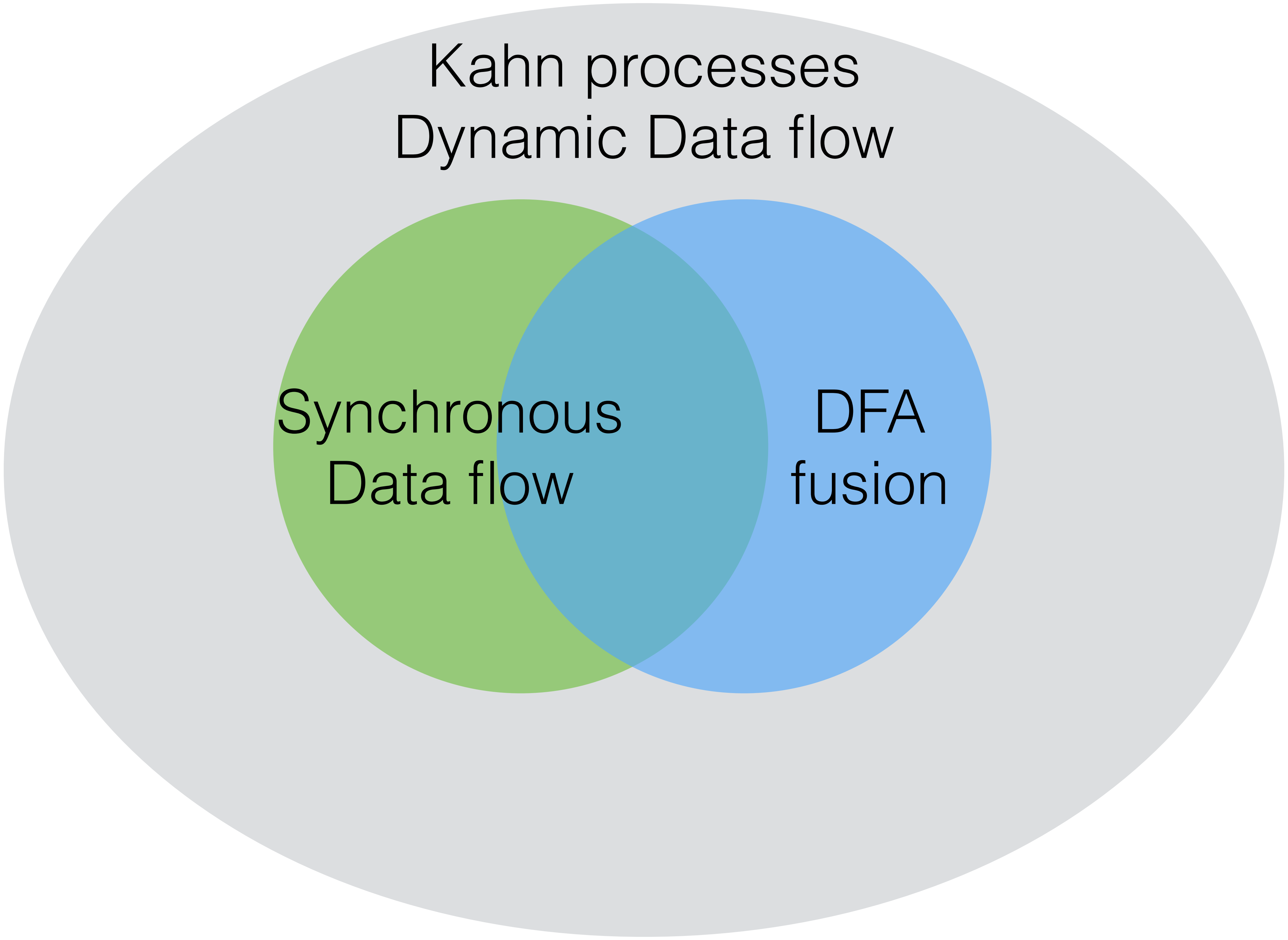


zip (filter (>0) xs) xs''



Deadlock!

Relationships

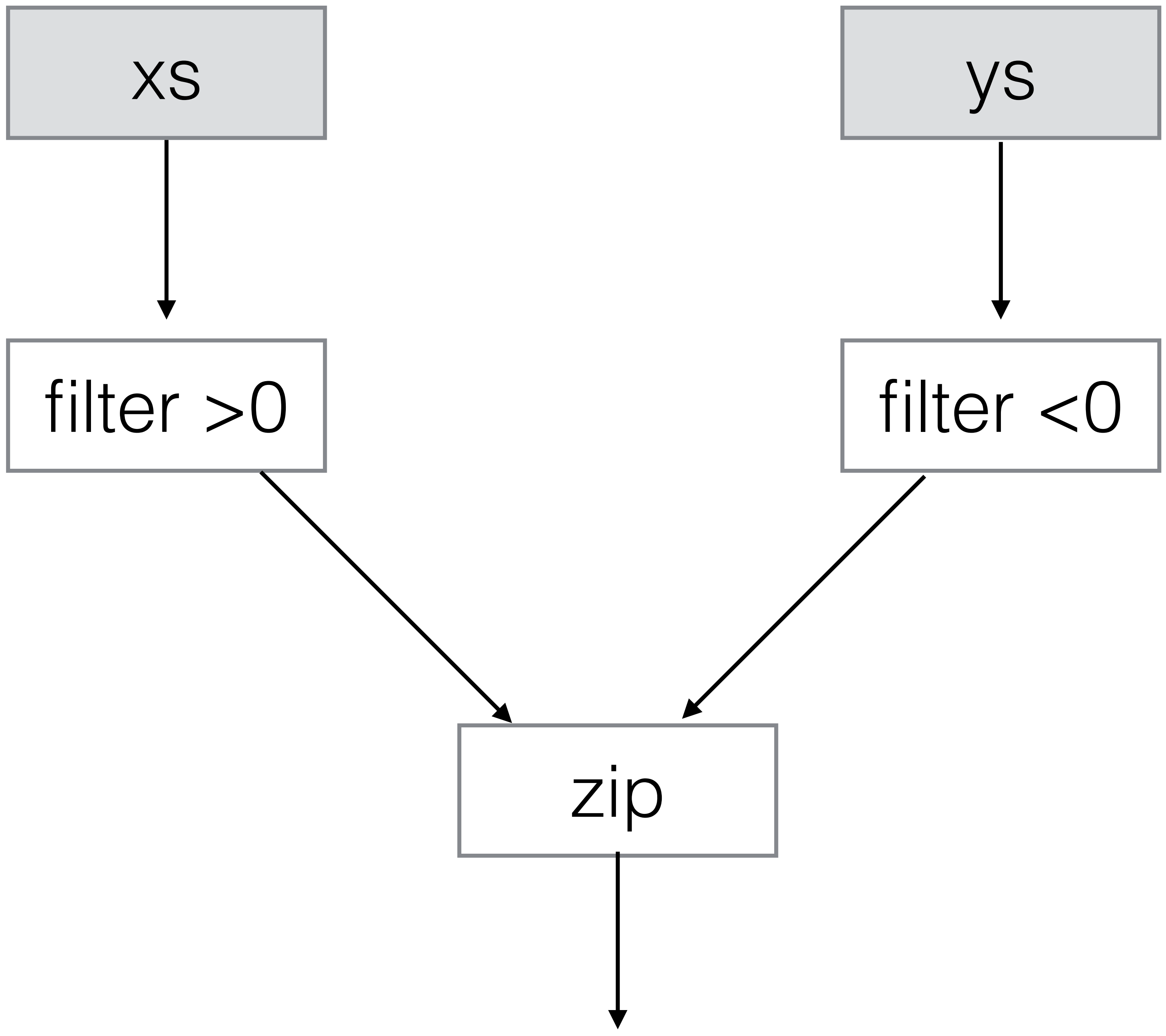


Next steps

(Image: Church of Subgenius)

Extend to multiple machines

Order of fusion matters



Finish writing paper

$$\frac{s : \texttt{Pull}\ n \quad s \xrightarrow{\texttt{Some}\ n} t \quad \texttt{Value}\ n \notin a \quad \texttt{Finished}\ n \notin a}{\Gamma, s : a \mid t \vdash a \cup \{\texttt{Value}\ n\}} \qquad \frac{s : \texttt{Pull}\ n \quad s \xrightarrow{\texttt{None}} t \quad \texttt{Value}\ n \notin a \quad \texttt{Finished}\ n \notin a}{\Gamma, s : a \mid t \vdash a \cup \{\texttt{Finished}\ n\}} \quad (\text{Pull})$$

$$\frac{s : \texttt{Release}\ n \quad s \Rightarrow t \quad \texttt{Value}\ n \in a}{\Gamma, s : a \mid t \vdash a \setminus \{\texttt{Value}\ n\}} \quad (\text{Release}) \qquad \frac{s : \texttt{Close}\ n \quad s \Rightarrow t \quad \texttt{Value}\ n \notin a}{\Gamma, s : a \mid t \vdash a \cup \{\texttt{Finished}\ n\}} \quad (\text{Close})$$

$$\frac{s : \texttt{Update}\ f\ n \quad s \Rightarrow t \quad \mathit{fvs}(f) \subset a}{\Gamma, s : a \mid t \vdash a} \quad (\text{Update}) \qquad \frac{s : \texttt{If}\ f\ n \quad s \Rightarrow t \quad \mathit{fvs}(f) \subset a}{\Gamma, s : a \mid t \vdash a} \quad (\text{If})$$

$$\frac{s : \texttt{Out}\ f\ n \quad s \Rightarrow t \quad \mathit{fvs}(f) \subset a}{\Gamma, s : a \mid t \vdash a \cup \{\texttt{Value}\ n\}} \quad (\text{Out}) \qquad \frac{s : \texttt{OutFinished}\ n \quad s \Rightarrow t}{\Gamma, s : a \mid t \vdash a \cup \{\texttt{Finished}\ n\}} \quad (\text{OutFinished})$$

$$\frac{s : \texttt{Skip} \quad s \Rightarrow t}{\Gamma, s : a \mid t \vdash a} \quad (\text{Skip})$$

$$\frac{s : \texttt{Done} \quad s \Rightarrow t \quad \forall c \in \mathbf{inputs}\ m \cup \mathbf{outputs}\ m.\ \texttt{Finished}\ c \in a}{\Gamma, s : a \mid t \vdash a} \quad (\text{Done})$$

$$\frac{\Gamma \mid p \vdash a \quad \Gamma \mid q \vdash b \quad \forall n \in (a \setminus b) \cup (b \setminus a).\ n \in \mathbf{outputs}\ m}{\Gamma, s : a \mid t \vdash a \cap b} \quad (\text{Subtype})$$

Figure 1. Invariant checking for machines

Prove preservation & progress

$$\forall a\, b.\ a\ \texttt{ok}\ \wedge\ b\ \texttt{ok}\ \wedge \forall c.\ \texttt{merge}\ a\ b = c \implies c\ \texttt{ok}$$

$$\forall a\, b.\ a\ \texttt{ok}\ \wedge\ b\ \texttt{ok}\ \wedge \forall c.\ \texttt{mergeV}\ a\ b = c \implies c\ \texttt{ok}$$

$$\begin{array}{ll} \forall a\, b. & a\ \texttt{ok}\ \wedge\ b\ \texttt{ok} \\ & \wedge\ \texttt{inputs}\ a \cap \texttt{inputs}\ b = \emptyset\ \wedge\ \texttt{outputs}\ a \cap \texttt{outputs}\ b = \emptyset \\ & \wedge\ (\exists n.\ (\texttt{inputs}\ a \cap \texttt{outputs}\ b) \cup (\texttt{outputs}\ a \cap \texttt{inputs}\ b) = \{n\}) \\ \implies & \exists c.\ \texttt{mergeV}\ a\ b = c \end{array}$$

Conclusion

- Paper published since last review
- Have prototype & working on another paper