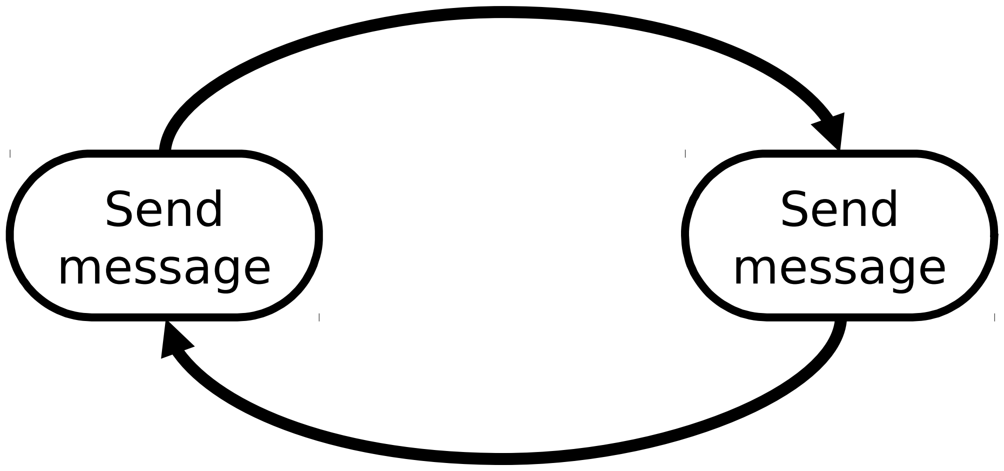
Solving CSP

Deadlock and Livelock

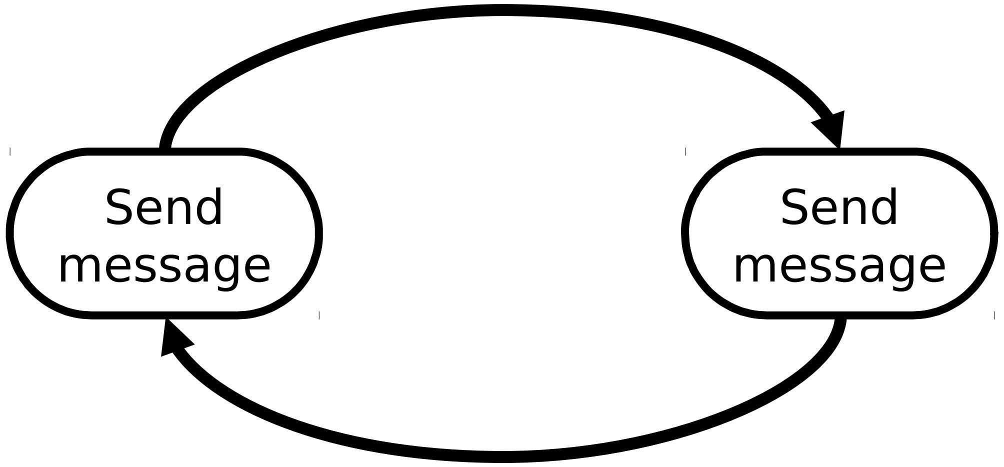
- Deadlock and livelock prevent the system continuing by starving its resources. It can be seen as a total system failure.
- Deadlock occurs when each process is locked out by another.
- Livelock occurs when each process tries to let another go first.
- There are ways to recover from deadlock and livelock, but the best solution is to avoid it altogether

Deadlock



Deadlock

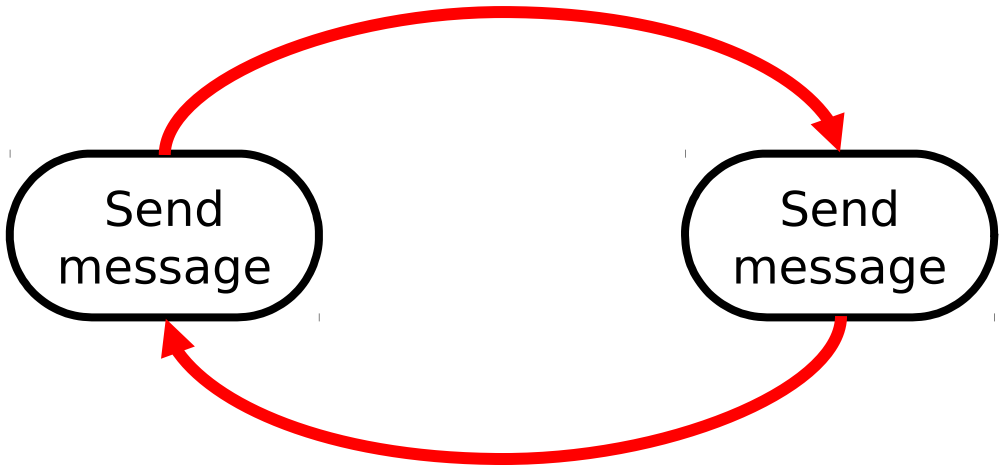
I want
to go
first!



I want
to go
first!

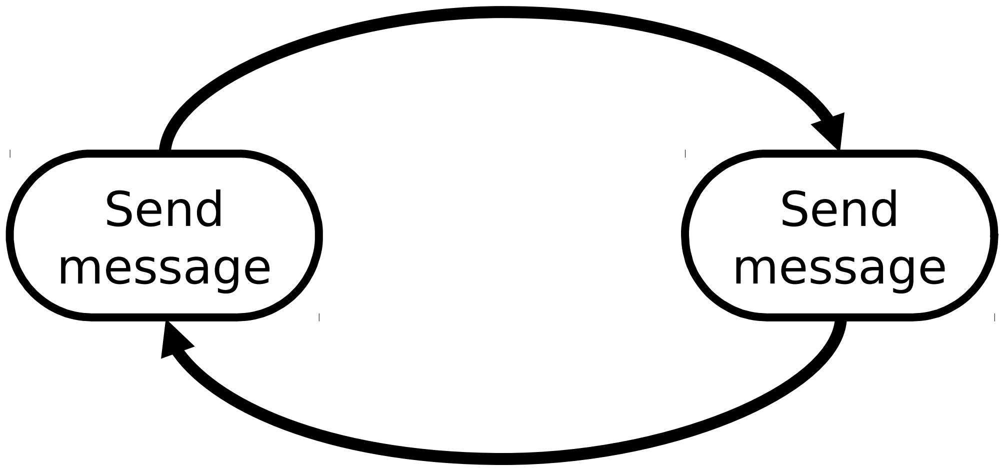
Deadlock

I want
to go
first!



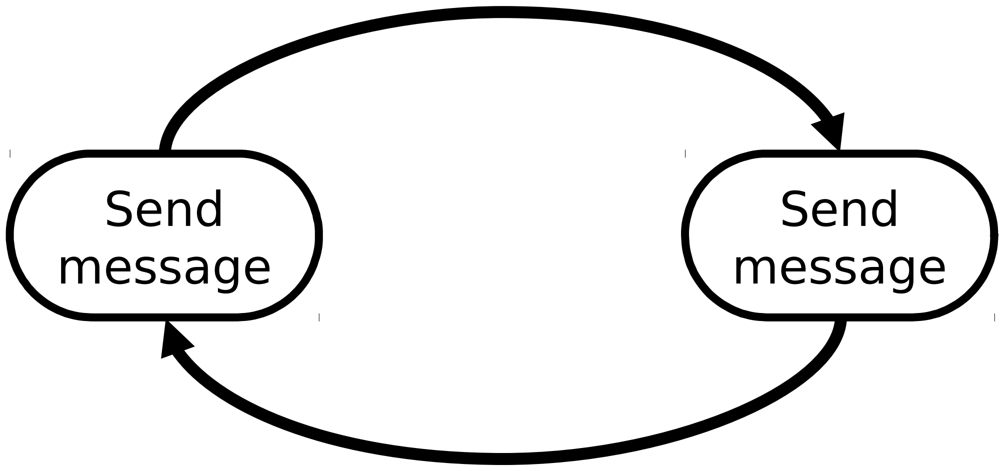
I want
to go
first!

Livelock



Livelock

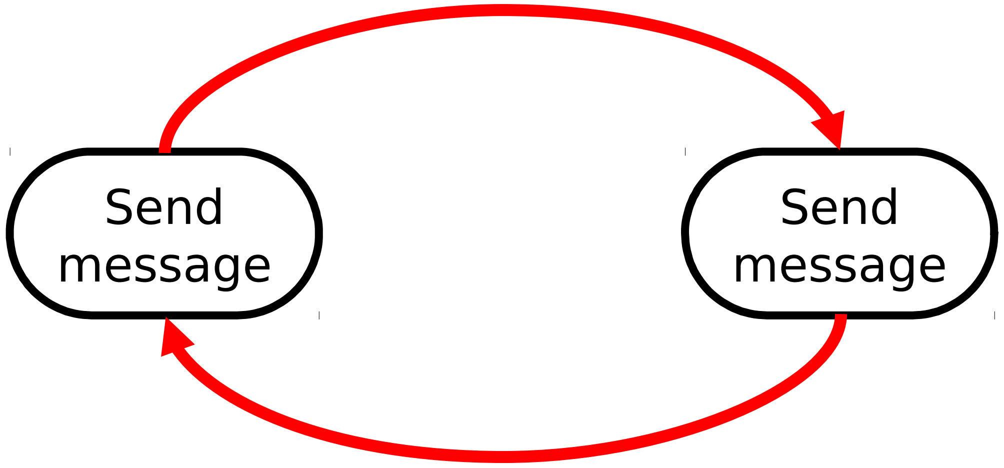
You go first, I insist



You go first, I insist

Livelock

You go first, I insist



You go first, I insist

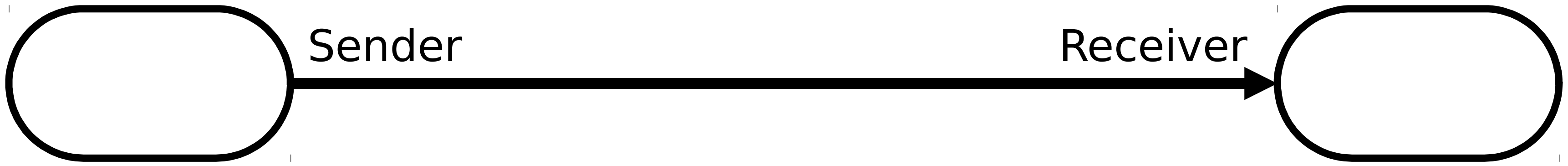
Senders and Receivers

- We can define two types of processes; Senders and Receivers.

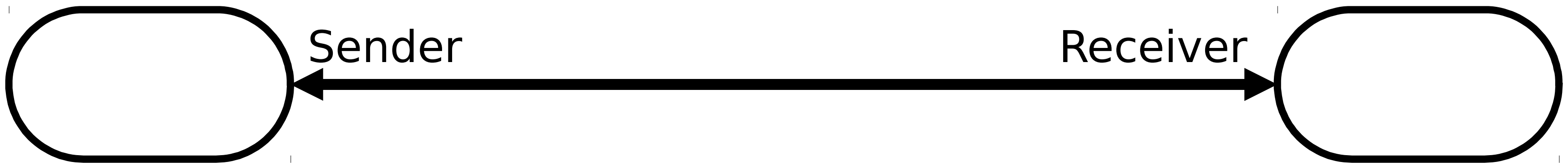
- Senders generate new messages and send them to receivers. They will wait for a response in a finite amount of time, if one is expected.

- Receivers will always wait to receive messages, and will always generate and send a response in a finite amount of time, if one is expected.

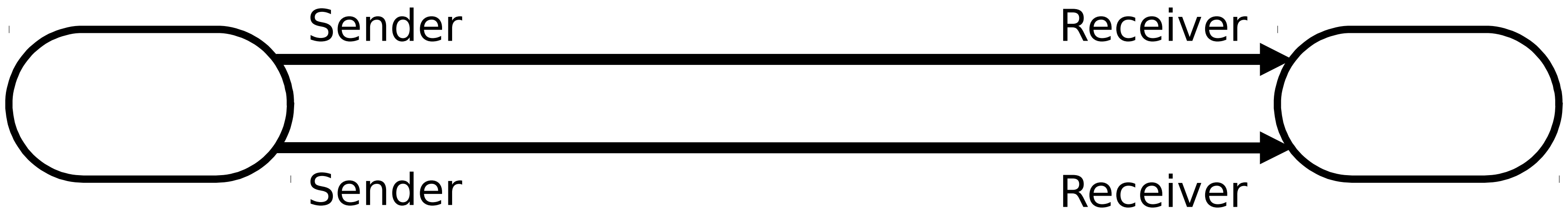
Senders and Receivers



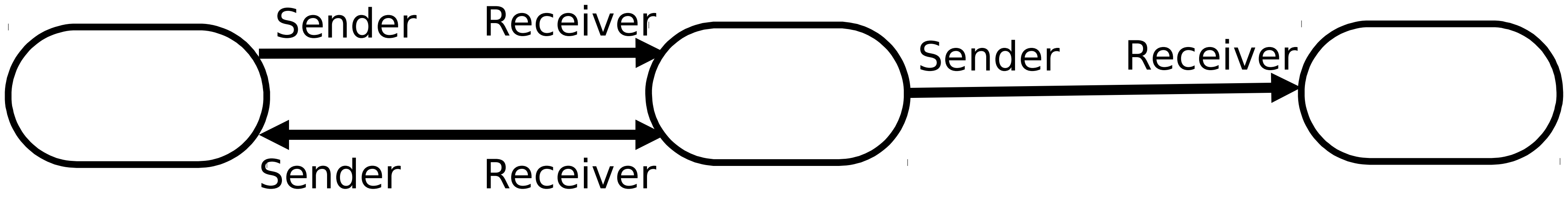
Senders and Receivers



Senders and Receivers



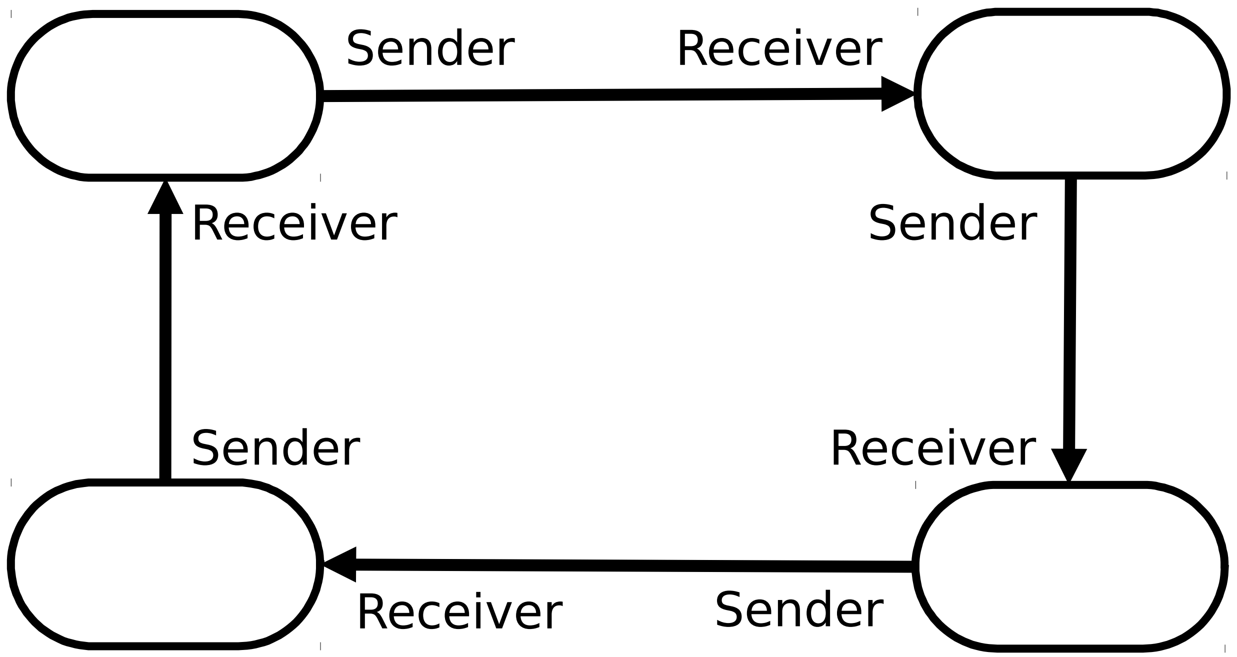
Senders and Receivers



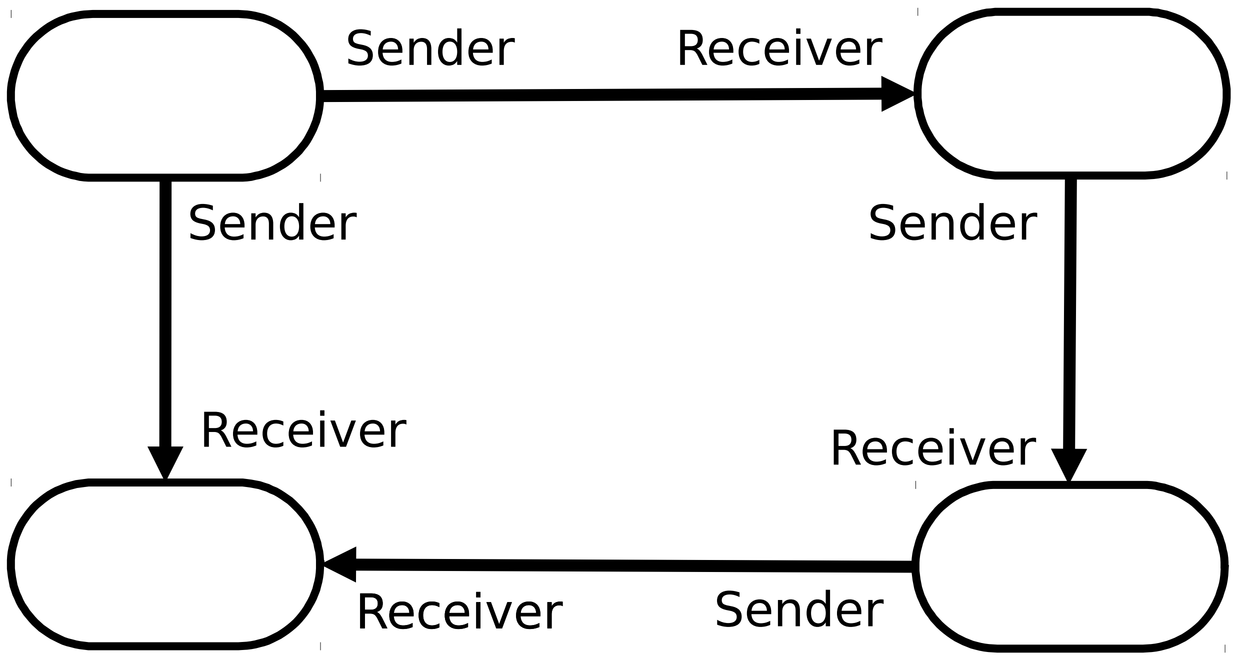
Senders and Receivers

- Senders and Receivers allow us to avoid deadlock.
- As long as no Senders and Receivers interact in a loop, deadlock cannot occur.
- Livelock *might* still be a problem but its actually quite hard to get that to occur without trying to (famous last words…).

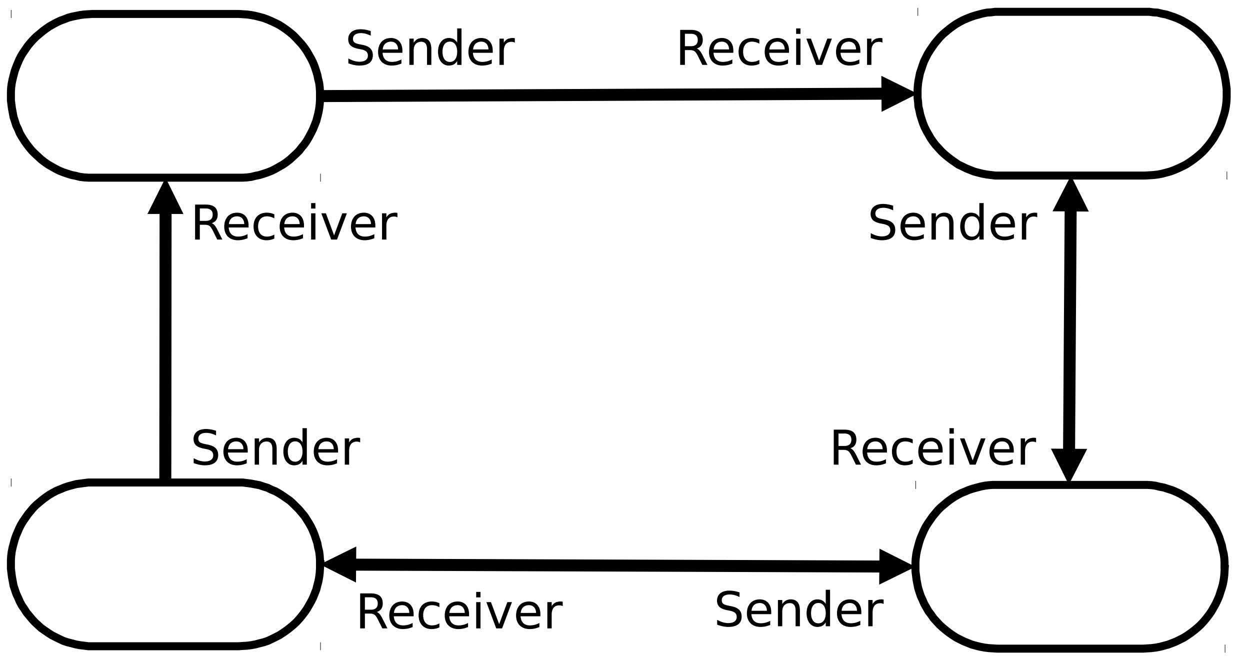
Senders and Receivers



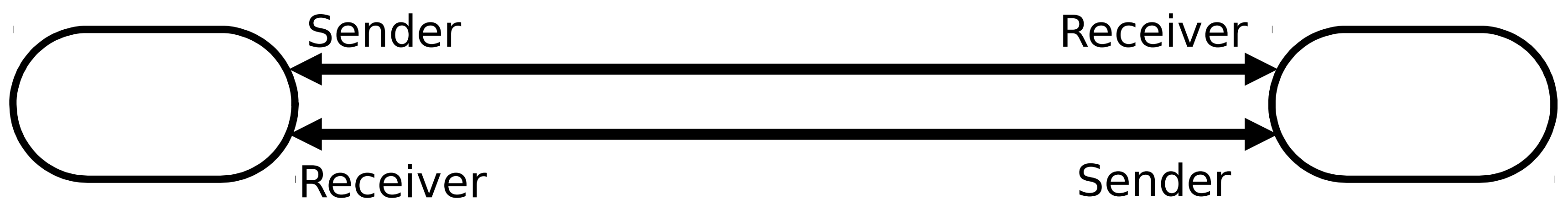
Senders and Receivers



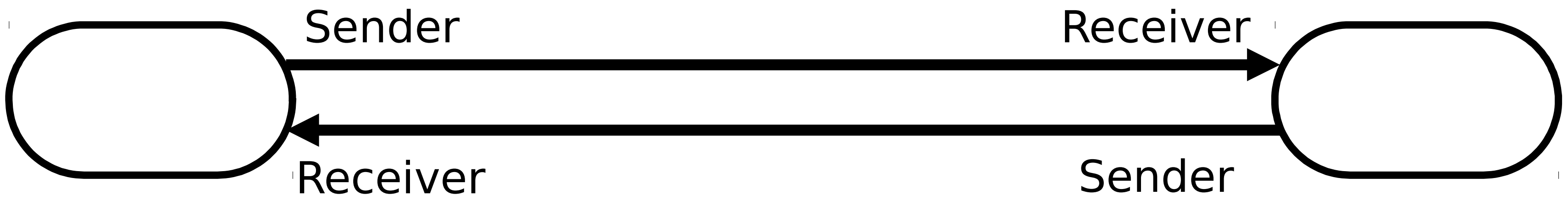
Senders and Receivers



Senders and Receivers



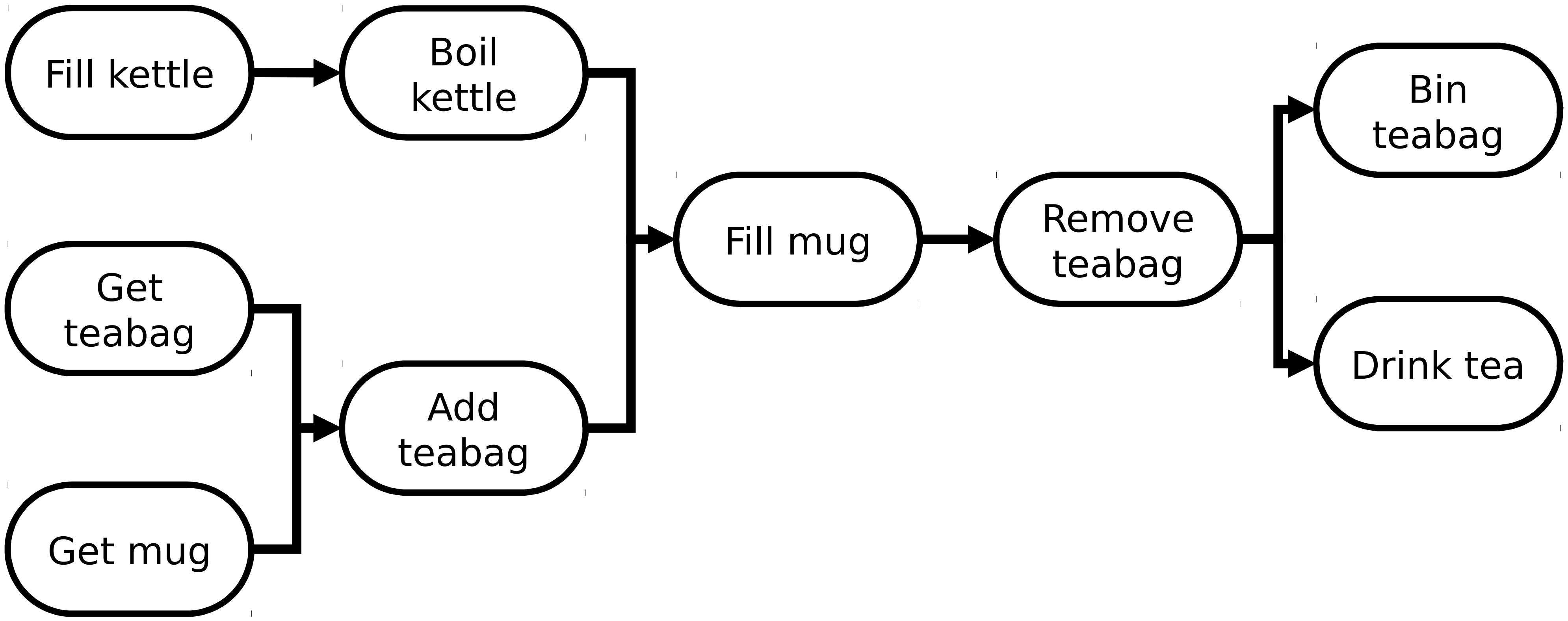
Senders and Receivers



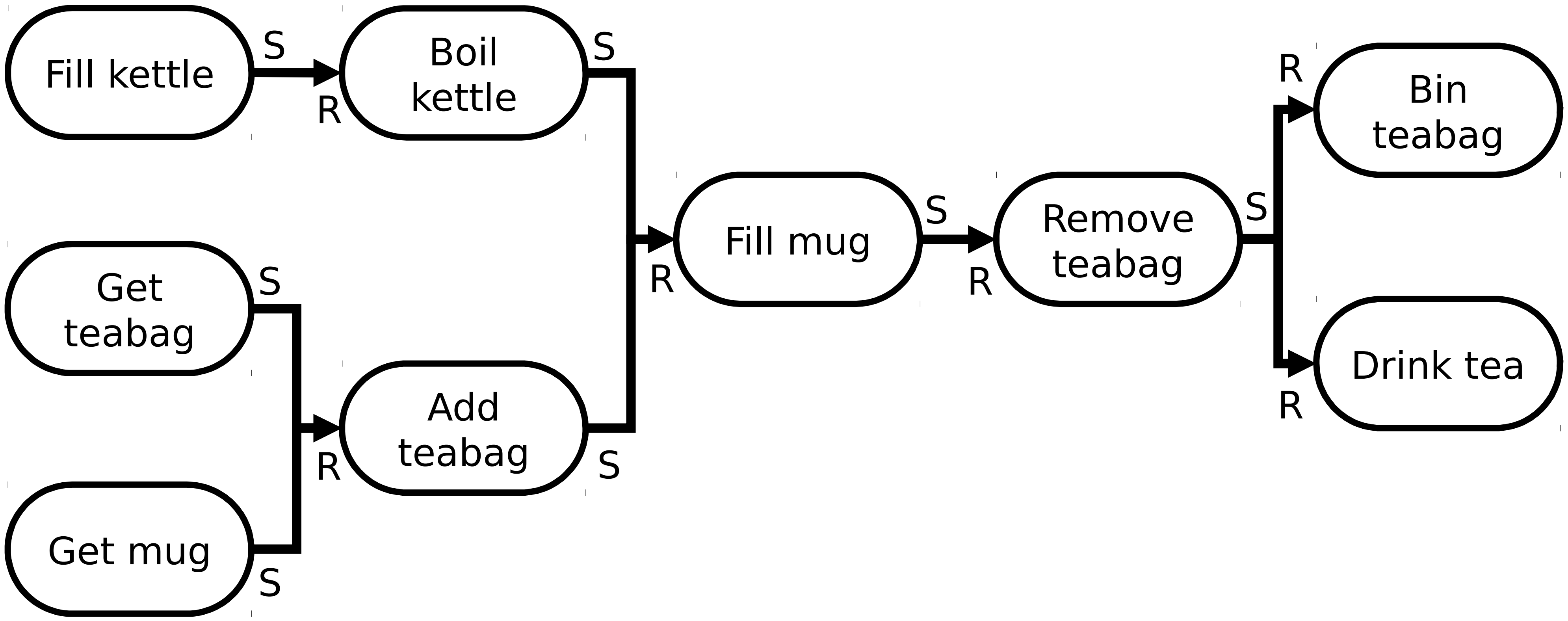
Senders and Receivers

- The Sender Receiver model solves deadlock.
- But only if you’ve implemented your system as designed.
- Still possible to livelock.
- Still possible to make any other mistake.
- Still possible to avoid deadlock with careful management.
- Note that it may be referred to as the Client Server model in literature.

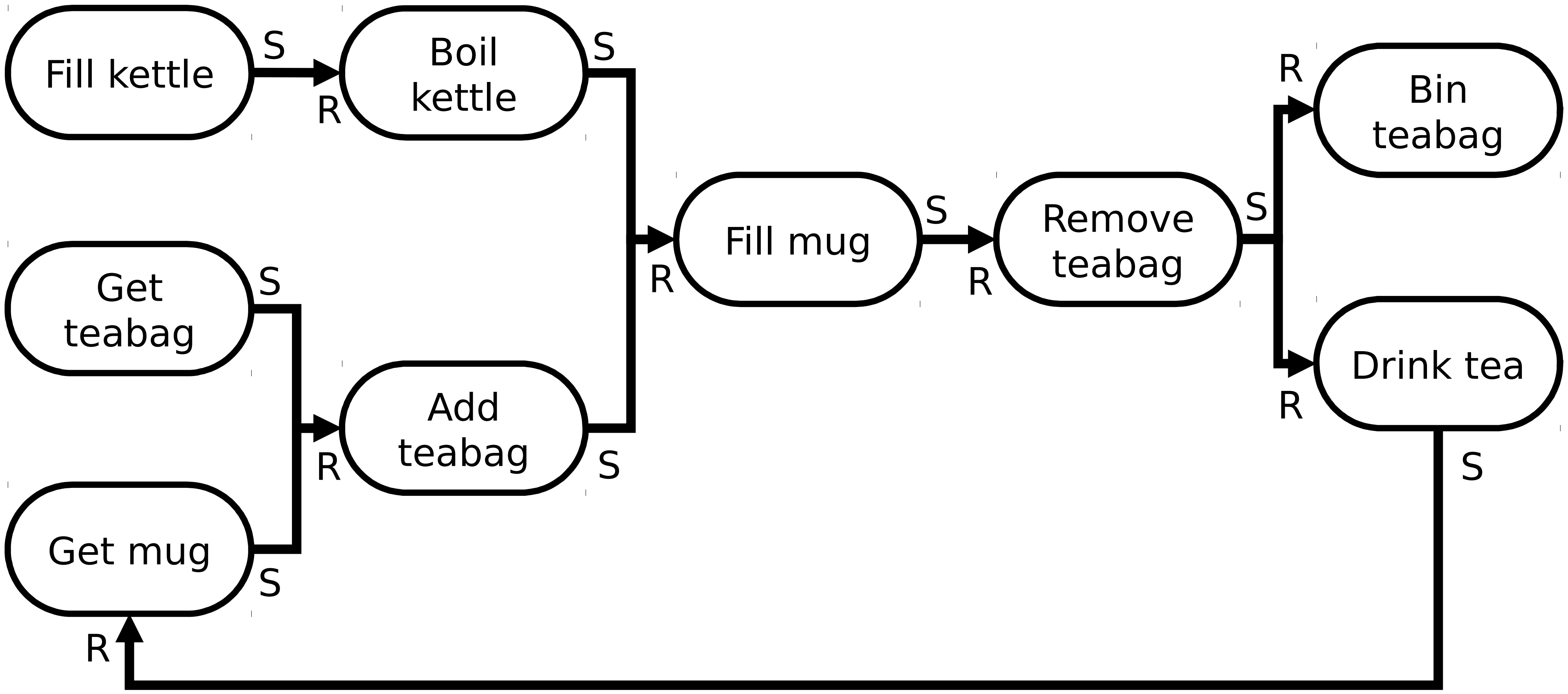
Senders and Receivers



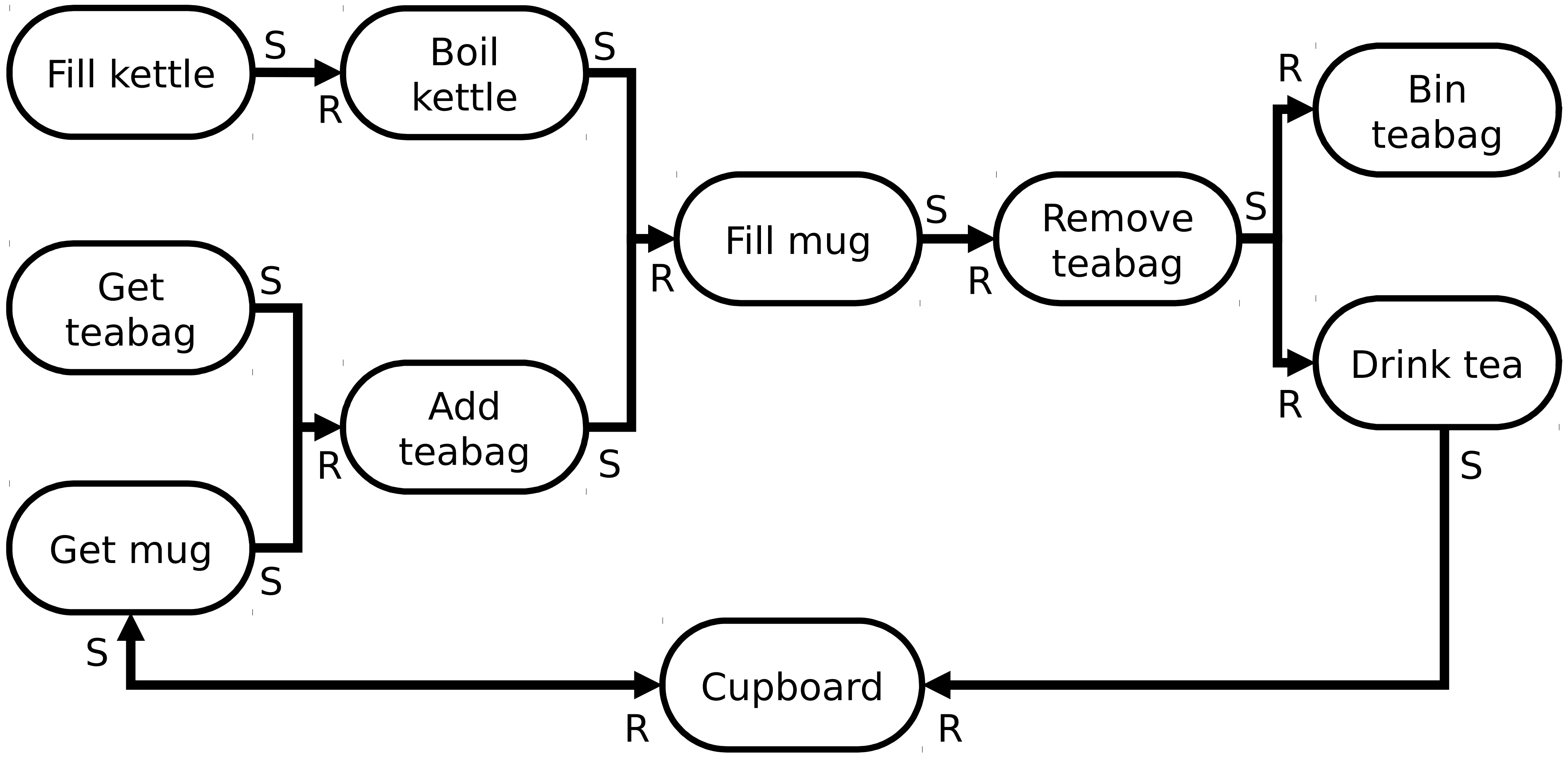
Senders and Receivers



Senders and Receivers



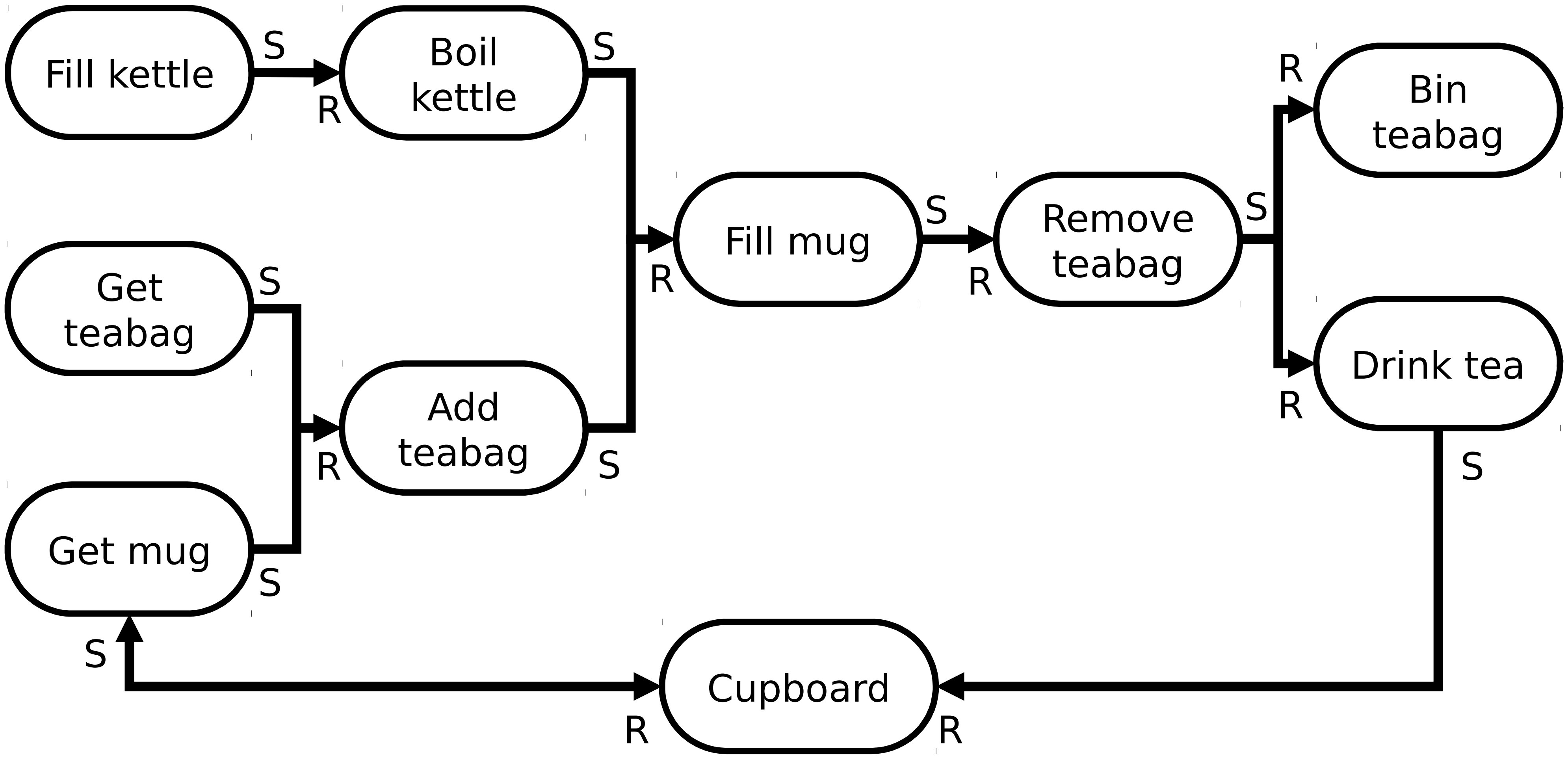
Senders and Receivers



The Resting Point

- It can help when checking the Sender/Receiver relationship to think of each process as having a Resting Point.
- This is the point at which the process will rest, waiting for an interaction and can be either listening for input, or waiting to send output.
- This may change as the process changes state and is intended only as a guide.

The Resting Point



Buffers

- The Cupboard process is acting as a *buffer.*
- A buffer is a place where messages can be stored to give the system some extra capacity.
- We can use this to swap the resting point between getting a mug and drinking tea

A Note About Bi-directionality

- If we passed a channel to a process rather than one end, then we could read or writer in either process.
- We have 1 channel, and can communicate in either direction.
- SPOT THE PROBLEM WITH THIS.