Distributed Networks

Networked Communication

- Communication over a distributed networks is different to what we've been doing so far.
- We need to dynamically create connections between processes at runtime.
- We can do this on local machines as well, but there's not much point in doing so if it can be avoided.

Dynamic Creation

Dynamic Creation

```
@process
def receiver():
    receivers_channel = Channel('A')
    print(receivers_channel.address)
    reader = receivers_channel.reader()
    while True:
        message = reader()
        print(message)

if __name__ == '__main__':
    Parallel(receiver())
```

Dynamic Creation

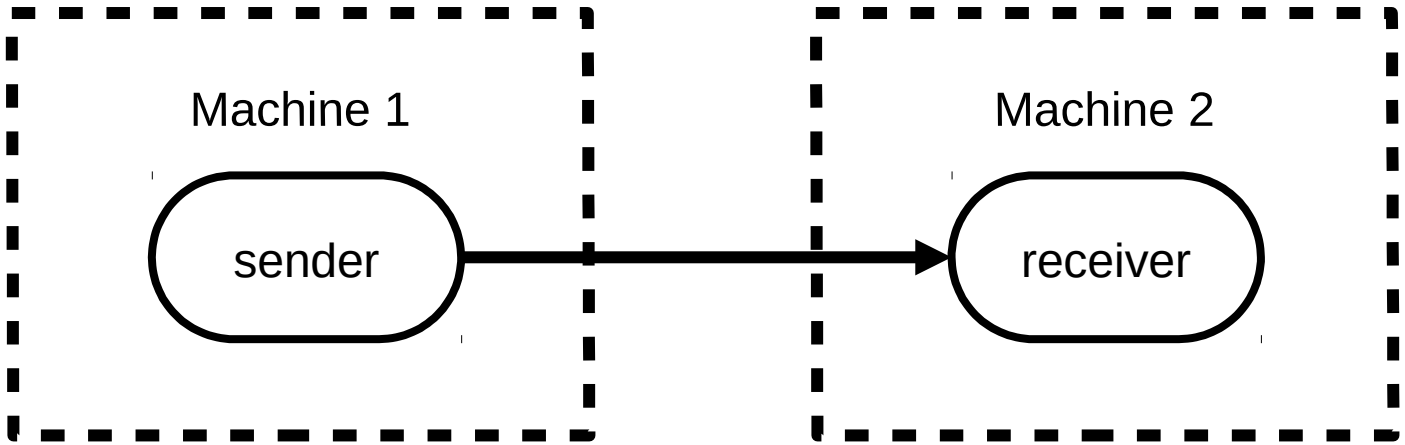
```
@process
def sender():
    receiver_ip = raw_input("Receiver IP? :")
    receiver_port = int(raw_input("Receiver Port? :"))
    tuple = (receiver_ip, receiver_port)
    sender_channel = Channel(name='A', connect=tuple)
    writer = sender_channel.writer()
    writer("hello across a dynamic connection")

if __name__ == '__main__':
    Parallel(sender())
```

Dynamic Creation

- It works (Hopefully).
- Remember that you can only connect to a channel if it was given a name.
- So why won’t this work over a distributed network?
- As an aside, how many channels did we create in the previous example?

Dynamic Network Creation



Dynamic Network Creation

- We’re going to need a new type of process, @multiprocess
- A multiprocess is very much like a process except it is built on top of Pythons multiprocessing module.
- The key difference is that we can define its host address.
- For this part to work we need to use Python 2.6 or above, but not Python 3 (Probably).

Dynamic Network Creation

```
@multiprocess(
        pycsp_host='172.24.5.60',
        pycsp_port=10000)
def receiver():
    channel_in = Channel('A')
    print(channel_in.address)
    reader = channel_in.reader()

    while True:
        message = reader()
        print(message)

if __name__ == '__main__':
    Parallel(receiver())
```

Dynamic Network Creation

```
@multiprocess
def sender():
    to_receiver_channel = Channel('A',
        connect=('172.24.5.60', 10000))

    writer = to_receiver_channel.writer()
    print("ready to send")
    writer("hello over a network")
    print("send complete")

if __name__ == '__main__':
    Parallel(sender())
```

Network Problems

- Communication is usually very slow.
- Messages can take a long time to propagate through a network.
- The network is unreliable.
- Global memory doesn’t exist (yay!).
- Global time doesn’t exist (boo!).

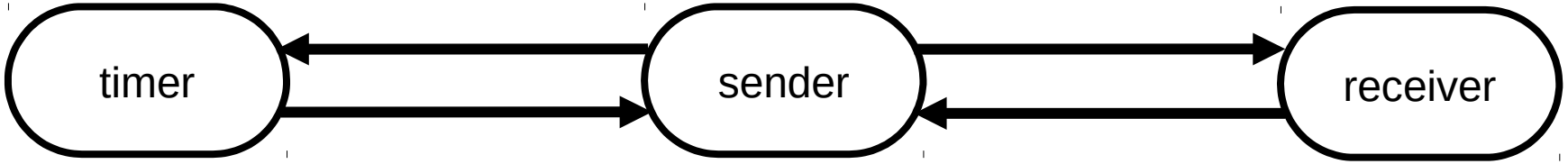
Network Problems

- We can help mitigate some of these problems by:
- Minimising the amount of communication between network processes.
- Minimising the amount of processes, communications and states within a system.
- Not using any global time, and assuming everything is out of order from the off.

Network Problems

- Timers might just be our best friend in Distributed Systems.
- If we send a message over a network we want to know it has gotten there.
- The Receiver sends a response to the Sender.
- During sending the Sender starts a timer process. This will send a notification to the Sender after a given time.
- If the timer sends before the response, then we’ve probably deadlocked.

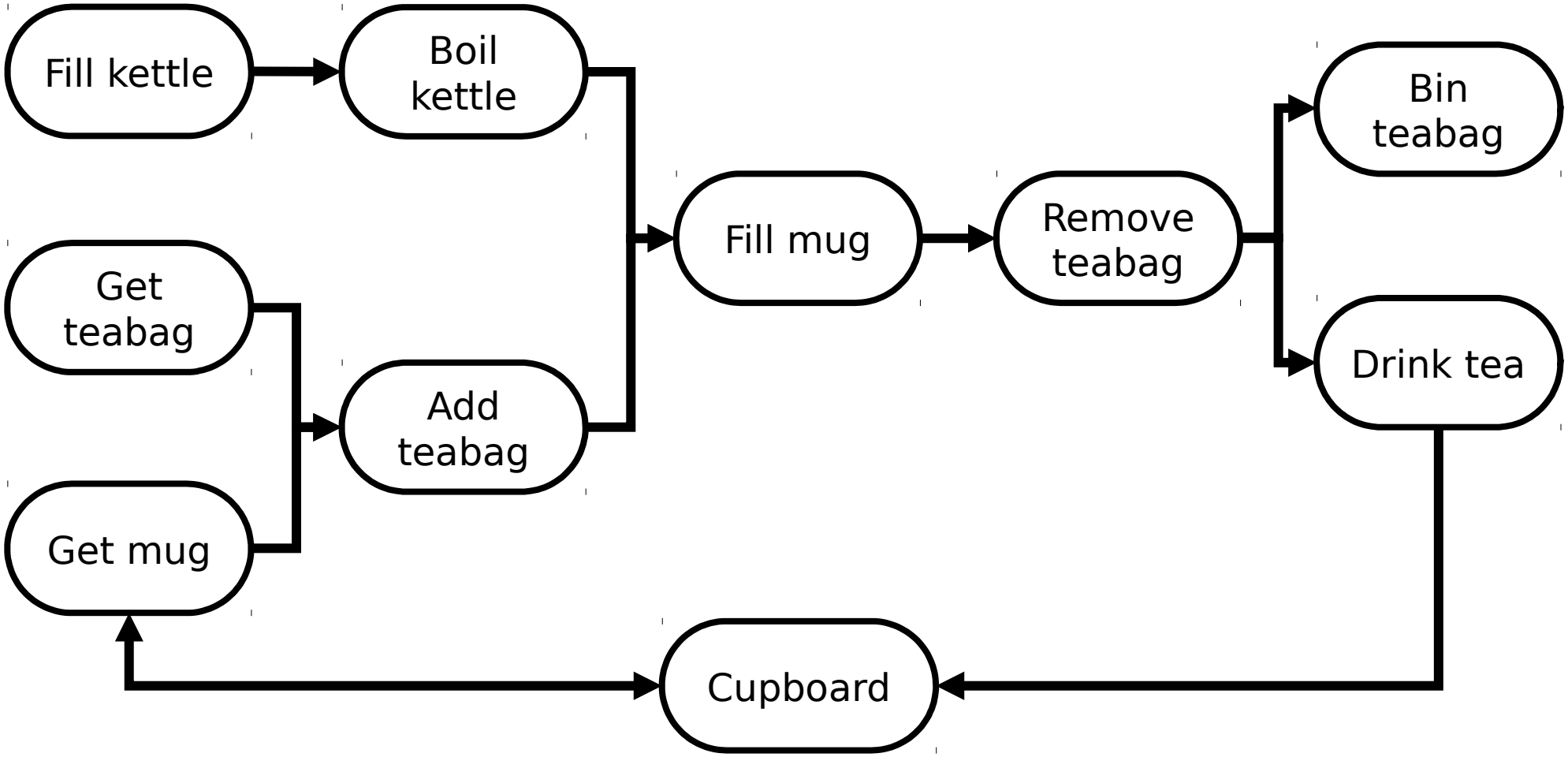
Network Problems



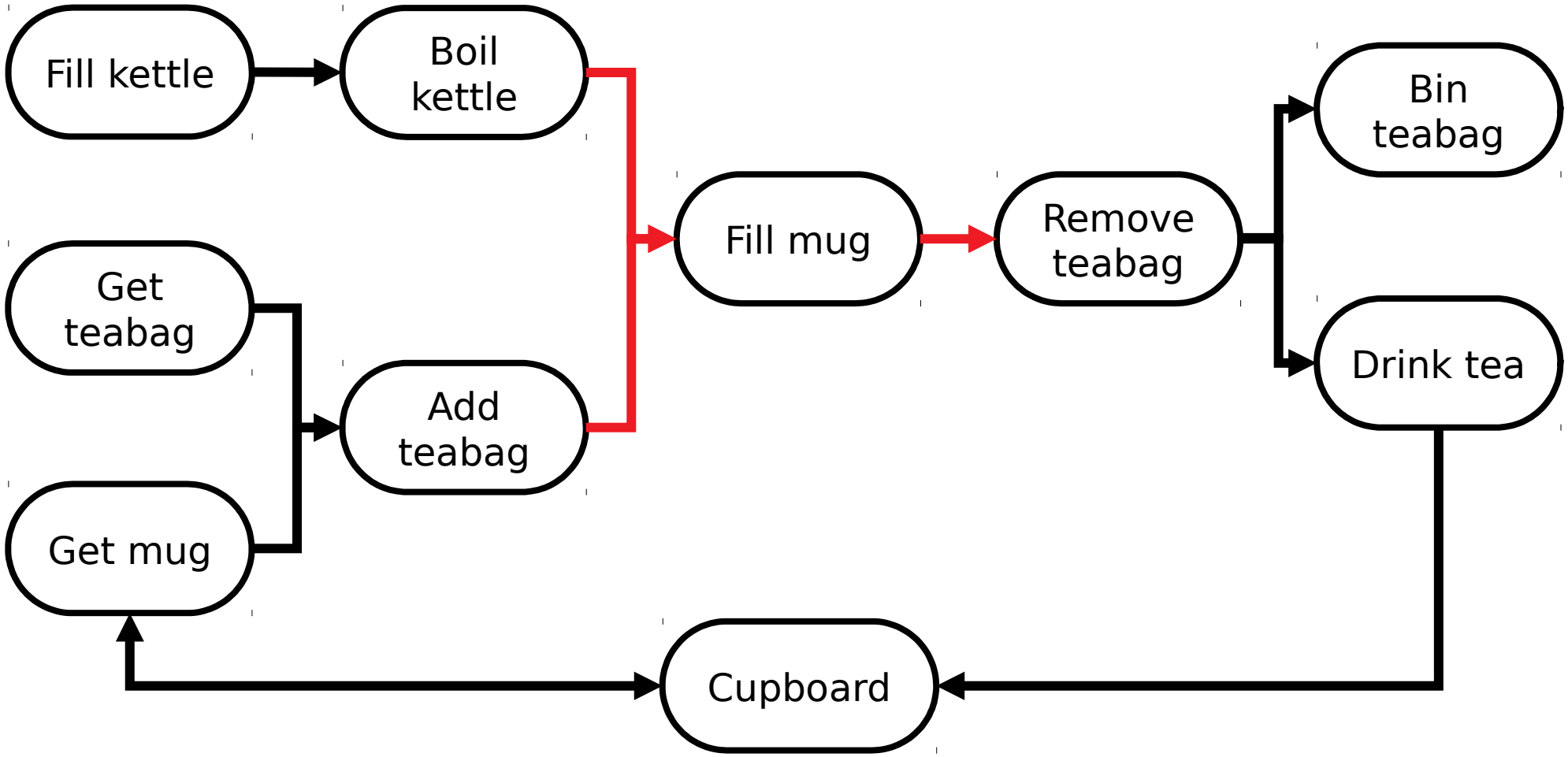
Poisoning

- To ‘help’ shutdown system, CSP defines Poisoning
- Any channel can be poisoned. Once poisoned, it will throw an Exception any time a process tries to send a message over it.
- The idea is that Poisoning will propagate over a network, shutting down everything.

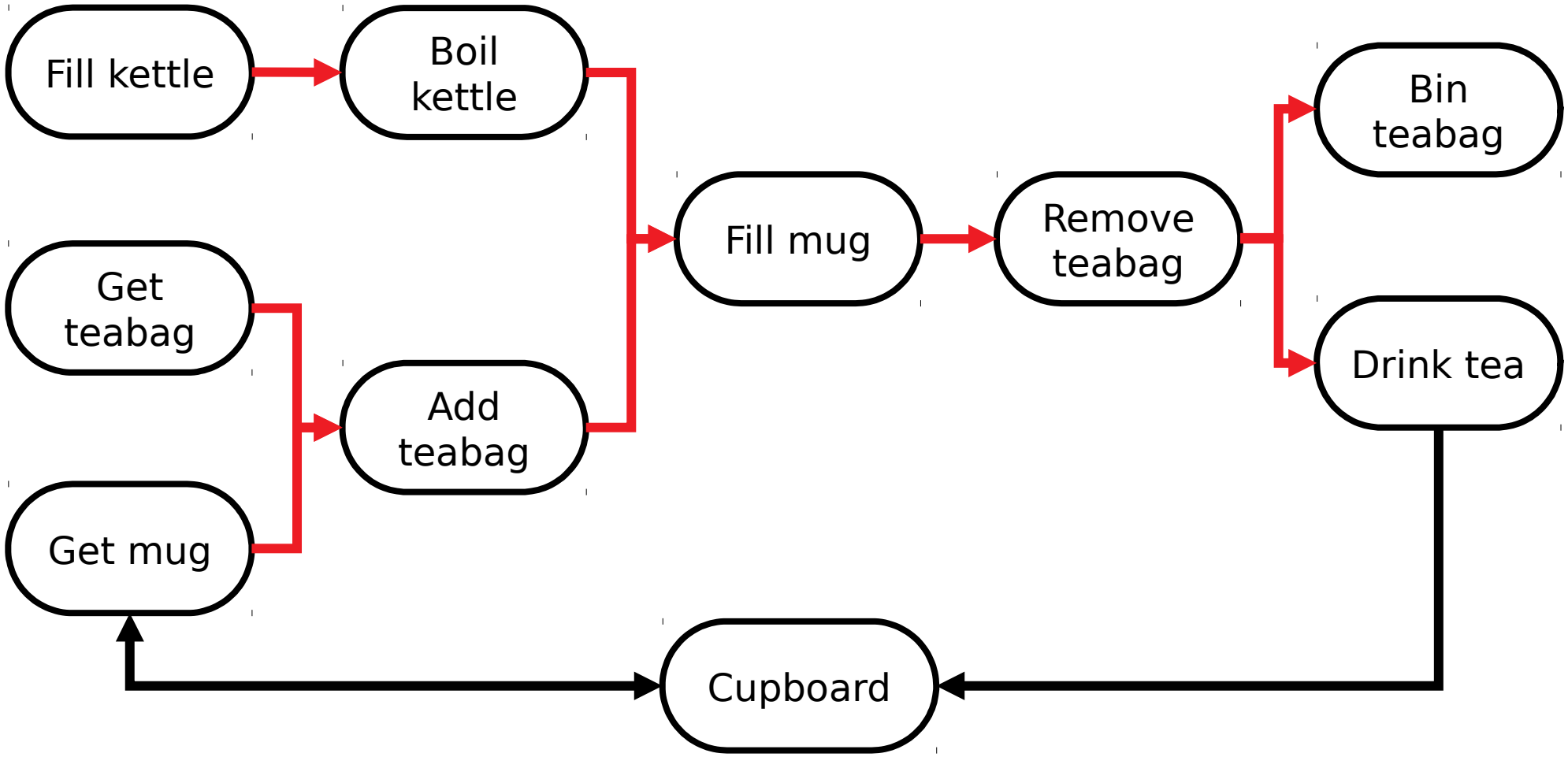
Poisoning


Fill kettle
Boil kettle
Get teabag
Add teabag
Get mug
Fill mug
Remove teabag
Bin teabag
Drink tea
Cupboard

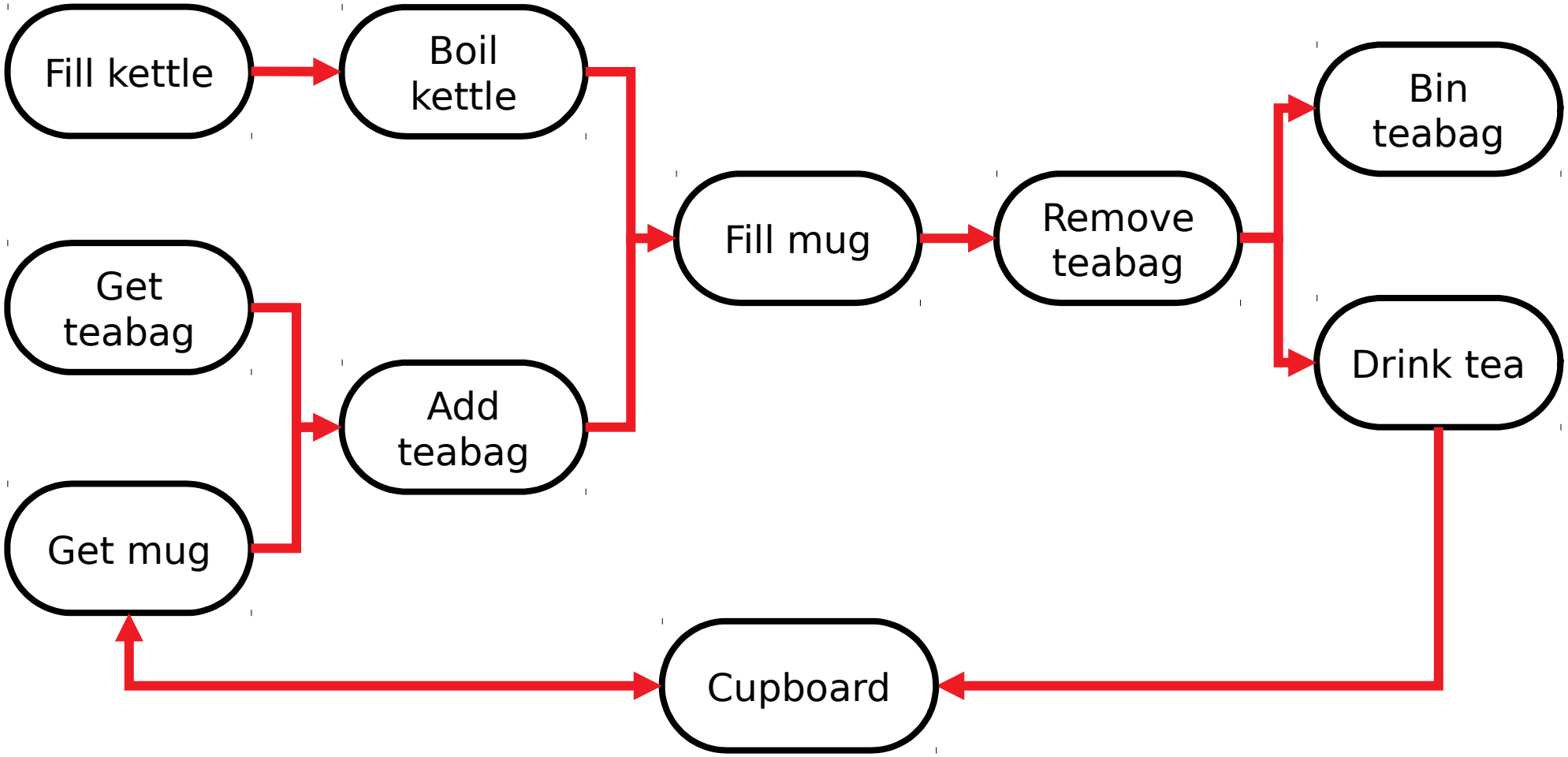
Poisoning


Fill kettle
Boil kettle
Get teabag
Add teabag
Get mug
Fill mug
Remove teabag
Bin teabag
Drink tea
Cupboard

Poisoning


Fill kettle
Boil kettle
Get teabag
Add teabag
Get mug
Fill mug
Remove teabag
Bin teabag
Drink tea
Cupboard

Poisoning


Fill kettle
Boil kettle
Get teabag
Add teabag
Get mug
Fill mug
Remove teabag
Bin teabag
Drink tea
Cupboard

Poisoning

- Poisoning relies on throwing lots of exceptions.
- You might find this to be perfectly acceptable.
- I prefer other methods, like more careful management of interactions.
- This is a stylistic choice though.