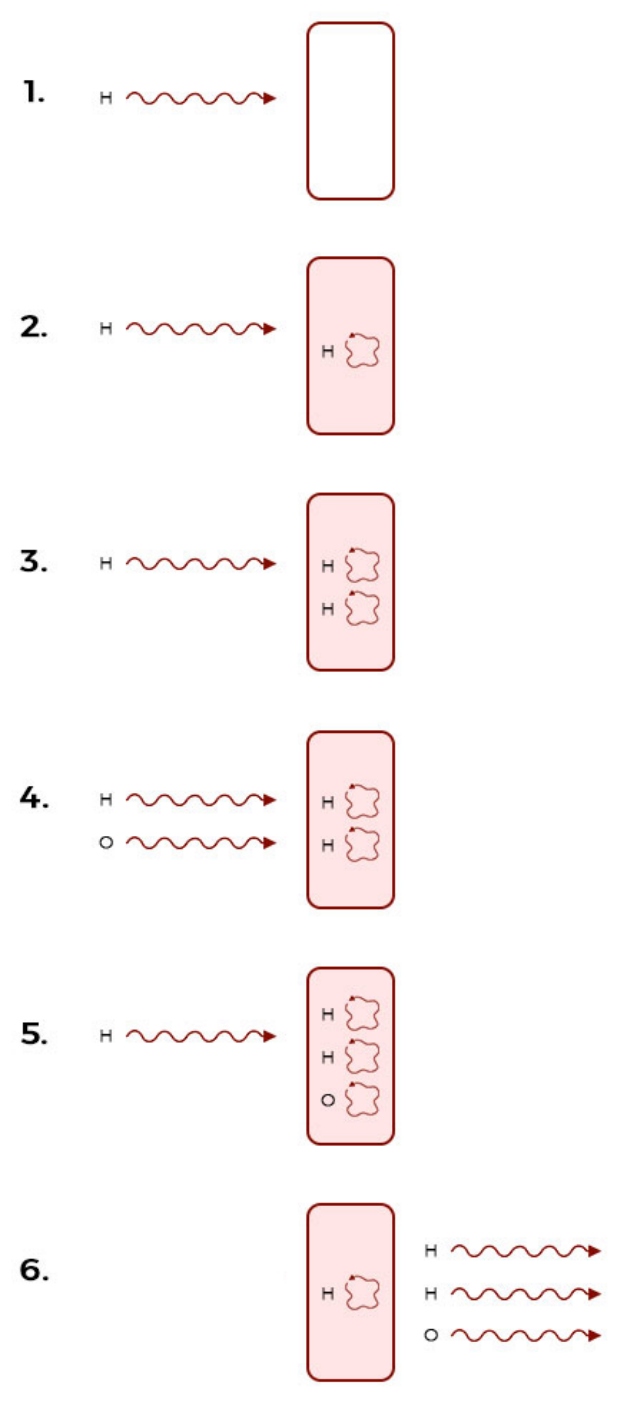
Build a Molecule

This problem simulates the creation of water molecule by grouping three threads representing Hydrogen and Oxygen atoms.

Problem

Suppose we have a machine that creates molecules by combining atoms. We are creating water molecules by joining one oxygen and two hydrogen atoms. The atoms are represented by threads. The machine will wait for the required atoms (threads), then group one oxygen and two hydrogen threads to simulate the creation of a molecule. The molecule then exists the machine. You have to ensure that one molecule is completed before moving onto the next molecule. If more than the required number of threads arrive, they will have to wait. The figure below explains the working of our machine:



Two Hydrogen threads are admitted in the machine as they arrive but when the third thread arrives in step 3, it is made to wait. When an Oxygen thread arrives in step 4, it is allowed to enter the machine. A water molecule is formed in step 5 which exists the machine in step 6. That is when the waiting Hydrogen thread is notified and the process of creating more molecules continues. The threads can arrive in any order which means that HHO, OHH and HOH are all valid outputs.

The code for the class is as follows:

```
class H2OMachine
```

```
    def initialize
    end


    def HydrogenAtom
    end

    def OxygenAtom
    end
end
```

The input to the machine can be in any order. Your program should enforce a 2:1 ratio for Hydrogen and Oxygen threads, and stop more than the required number of threads from entering the machine.

Solution

Our molecule making machine is represented by the class `H2OMachine`, which contains two main methods; `HydrogenAtom` and `OxygenAtom`.

The problem is solved using basic synchronization tools like `mutex` and `condition variable` in Ruby. The class consists of 4 variables; `mutex` for synchronization, a string array `molecule` for holding 3 elements (atoms), `count` for current index of the molecule array, and `cond_var` for waiting on the mutex when no space is available in the array and broadcasting when space becomes available.

In the constructor, `count` is initialized with 0.

```
def initialize
        @molecule = []
        @count = 0
        @mutex = Mutex.new
        @cond_var = ConditionVariable.new
    end
```

For synchronization purpose, the entire logic of `HydrogenAtom` is wrapped in `mutex.synchronization`. First of all, we check the frequency of Hydrogen atom in the molecule by using `count()` on `molecule`. The

function checks the number of Hydrogen atoms in it. If the array has reached its capacity of 2 Hydrogen atoms, then the thread should wait for space in a new molecule. If the frequency is less than 2 it means space is available in the current molecule. Hence, H is placed in the array and `count` is incremented. So far, the code of `HydrogenAtom` is as follows:

```
def HydrogenAtom
    @mutex.synchronize do
        if (@molecule.count("H") == 2)
            @cond_var.wait(@mutex)
        end

        @molecule[@count] = "H"
        @count += 1
    end
end
```

In case `molecule` is full and `count` is 3, then print the `molecule` and exit the machine. The array `molecule` is reset and `count` goes back to 0 for a new `molecule` to be built. At the end of the method, the waiting threads (atoms) are notified using `broadcast()`. The complete code for `HydrogenAtom()` is given below:

```
def HydrogenAtom
    @mutex.synchronize do
        if (@molecule.count("H") == 2)
            @cond_var.wait(@mutex)
        end

        @molecule[@count] = "H"
        @count += 1

        if(@count == 3)
            @molecule.each do |atom|
                puts atom
            end

            @molecule = []
            @count = 0
        end
        @cond_var.broadcast
    end
end
```

The second method `OxygenAtom` is the same as `HydrogenAtom` with the only difference of the atom frequency check in the array molecule. If it contains an Oxygen atom already, then the calling thread waits for space in a new molecule. If the count of Oxygen atom is not equal to 1 in the `molecule`, then an Oxygen atom “O” is placed in the next available space. The complete code of `OxygenAtom` is shown below:

```
def OxygenAtom
    @mutex.synchronize do
        if (@molecule.count("O") == 1)
            @cond_var.wait(@mutex)
        end

        @molecule[@count] = "O"
        @count += 1

        if(@count == 3)
            @molecule.each do |atom|
                puts atom
            end
            @molecule = []
            @count = 0
        end
        @cond_var.broadcast
    end
end
```

The complete code for the solution is as follows:

```
class H2OMachine

    def initialize
        @molecule = []
        @count = 0
        @mutex = Mutex.new
        @cond_var = ConditionVariable.new
    end

    def HydrogenAtom
        @mutex.synchronize do
            if (@molecule.count("H") == 2)
                @cond_var.wait(@mutex)
```

```
            end

            @molecule[@count] = "H"
            @count += 1

            if(@count == 3)
                @molecule.each do |atom|
                    puts atom
                end
                @molecule = []
                @count = 0
            end
            @cond_var.broadcast
        end
    end

    def OxygenAtom
        @mutex.synchronize do
            if (@molecule.count("O") == 1)
                @cond_var.wait(@mutex)
            end

            @molecule[@count] = "O"
            @count += 1

            if(@count == 3)
                @molecule.each do |atom|
                    puts atom
                end
                @molecule = []
                @count = 0
            end
            @cond_var.broadcast
        end
    end
end
```

We will now be creating 4 threads in order to test our proposed solution. Same object of `H2OMachine` is passed to the 4 threads: **t1**, **t2**, **t3** and **t4**.

t1 and **t4** act as Hydrogen atoms trying to enter the machine where as **t2** and **t3** act as Oxygen atoms. It can be seen from the output that only 1 molecule of H2O exits the machine while the extra Oxygen atom is not utilized.

```
class H2OMachine

    def initialize
        @molecule = []
        @count = 0
        @mutex = Mutex.new
        @cond_var = ConditionVariable.new
    end

    def HydrogenAtom
        @mutex.synchronize do
            if (@molecule.count("H") == 2)
                @cond_var.wait(@mutex)
            end

            @molecule[@count] = "H"
            @count += 1

            if(@count == 3)
                @molecule.each do |atom|
                    print atom
                end
                @molecule = []
                @count = 0
            end
            @cond_var.broadcast
        end
    end

    def OxygenAtom
        @mutex.synchronize do
            if (@molecule.count("O") == 1)
                @cond_var.wait(@mutex)
            end

            @molecule[@count] = "O"
            @count += 1

            if(@count == 3)
                @molecule.each do |atom|
                    print atom
                end
                @molecule = []
                @count = 0
            end
            @cond_var.broadcast
        end
    end
end

molecule = H2OMachine.new()


t1 = Thread.new(molecule) do
        molecule.HydrogenAtom
    end
t2 = Thread.new(molecule) do
        molecule.OxygenAtom
```

```
    end
t3 = Thread.new(molecule) do
        molecule.OxygenAtom

    end
t4 = Thread.new(molecule) do
        molecule.HydrogenAtom
    end

threads = []

threads << t3
threads << t1
threads << t2
threads << t4

threads.each(&:join)
```