Reading File serially

(i.e reading orderly, earlier we are getting files randomly, but now f1, f2, f3).

$$\begin{tikzcd} & \text{Serially} \arrow[dl] \arrow[dr] & \\ \text{Callbacks} & & \text{Promises} \end{tikzcd}$$

Interview: How to execute serial function & how to execute a Parallel function.

Thru Callbacks

```
cb1()
{
        cb2()
}
cb2
{
        cb3()
}
cb
{
        c.log(data);
}
```

Calling Async fn. (callbacks) serially. Date ....

```
fs. readFile ('f1.txt', cb1);
function cb1 (error, data)
  {   if(error)
        { c.log (error);
        }
      else{
          c.log ("   " + data);
          fs. readFile ('f2.txt', cb2);
        }
  }
function cb2 (error, data)
  {   if(err)
        { c.log(error); }
      else{
          c. log("  " + data);
          fs. readFile ('f3.txt', cb3);
   }
function cb3( error, data)
  {   if(error)
        { c. log(error);
      else
        { c. log ("  " + data);
          ~~fs. readFile ('~~
        }
  }
```

Spiral

Serially

Date ....

Thru Promises

```
let fs= require('fs')
  c.log ('Before');
let f1p = fs.promises.readFile ('f1.txt');  (Pending).
f1p.then (cb1);  (resolve).
function cb1 (data){
    c.log (" File Data -> " + data);
    let f2p = fs.promises.readFile ('f2.txt'); (Pending)
    f2p.then (cb2); (resolve)
    }

function cb2 (data){
     c.log (" File Data -> " + data);
     let f3p= fs.promises.readFile ('f3.txt'); (Pending)
     f3p.then (cb3); (resolve).
     }

function cb3 (data) {
        c.log (" File Data -> " + data);
       }

   c.log ('After');
```

O/P -

```
Before
After
File Data -> this is file1
File Data -> this is file2
File Data -> this is file3
```

→ This order will never change, ∵ we are running it serially.

syntatic sugar

Date ....

```
flp.then(cb).then(cb2).then(cb3).catch(function(err)
  {
      c.log(err);
  }
```

(o/p is same, we just wrote it in a different way).