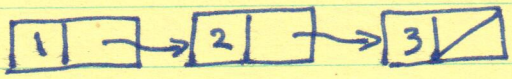
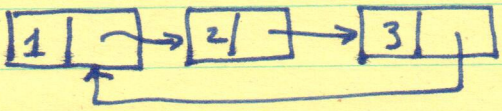
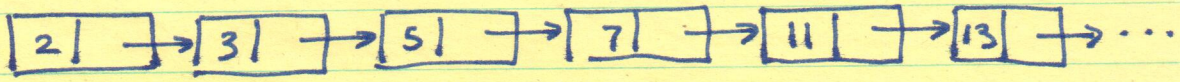
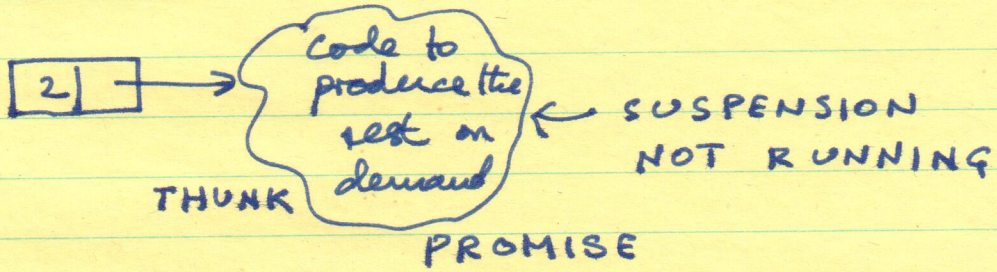






LAZY EVALUATION



We suspend a computation by wrapping it in a function

Wake up the code by applying the suspended code.

STREAMS : finite or infinite

Suppose there are finitely many primes : $p_1, \ldots p_k$

$$E = p_1 * p_2 * p_3 * \cdots * p_k + 1$$

$4n+1$: 13, 17, 29 . . .
$4n+3$: 7, 11, 31, . . .

```
let tlStr s =
  match s with
  | Eos -> failwith "..."
  | StrCons(x, t) -> t ()
```

```
let rec ones = StrCons(1, ones)
```

```
let rec ones = StrCons(1, fun () -> ones)
```

```
let rec nums_from n =
    StrCons(n, fun () -> nums_from (n+1))

let nats = nums_from 0
```

```
let fibstream =
  let rec fibgen a b =
      StrCons(a, fun () -> fibgen b (a+b))
  in
  fibgen 1 1;;
```

(2) (3) ~~4~~ (5) ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11
~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~
~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~ 31
. . .

```
let sift (p:int) : int stream -> int stream =
  filterStr (fun n -> n mod p <> 0)
```

```
let rec sieve (s: int stream) : int stream =
  match s with
  | Eos -> Eos
  | StrCons (p, g) ->
      StrCons (p, fun () ->
        sieve (sift p (g ())))

let primes = sieve (nums_from 2)

let rec zip s t =
  match s with
  | Eos -> t
  | StrCons (x, g) -> StrCons (x, fun () ->
      zip t (g ()))

let addStr s1 s2 =
  map2Str (fun x -> fun y -> x+y) s1 s2

let rec partial_sums s =
  match s with
  | Eos -> Eos
  | StrCons (h, t) -> StrCons (h,
    fun () -> (addStr (t ()) (partial_sums s)))
```

CORECURSION

COINDUCTION

```
let altnats = partial_sums ones
```