This is an implementation of the Sieve of Eartosthenes in psedu-code that allows new processes and channels to be added dynamically. Ben-Ari's pseudo-code only allows a fixed number of processes, e,g., a dining philosopher's program that begins

```
phil(i)         fork(i)
```

in parallel columns means there are 10 processes (i= 1..5) throughout.

Erlang allows dynamic process creation through "spawn". It does not have the "par" operator below, so all processes once spawned run in parallel, in a flat structure.

The par operator below is an implicit spawn, but also allows structure - so note how the channels passed as parameters to SIFT are passed on to FILTER and a new instance of SIFT.

As each new prime is discovered, a new FILTER process is added and SIFT moves to the right in chain that goes from, at the start,

```
INTEGERS(Q1) - Q1 - SIFT(Q1,Q2) - Q2 - OUTPUT(Q2)
```

to

```
INTEGERS(Q1) - Q1 - FILTER(Q1,Q2) - local Q - SIFT(Q,Q2) - Q2 - OUTPUT(Q2)
```

to

```
INTEGERS(Q1) - Q1 - FILTER(Q1,Q) - local Q - FILTER (Q,Q) - local Q -SIFT(Q,Q2) -
Q2 - OUTPUT(Q2)
```

Here the two local Q's are different, being created by different instances of SIFT.

```
Proc INTEGERS(chan QOUT)
   int N=1
   loop forever
      N++
      QOUT!N

Proc OUTPUT(chan QIN)
    int N
    loop forever
       QIN?N
       print(N)

Proc FILTER(int PRIME, chan QIN, chan QOUT)
    int N;
    loop forever
       QIN?N
       if (N MOD PRIME) =/= 0 then QOUT!N

Proc SIFT(chan QIN, QOUT)
  int PRIME
  chan Q
    QIN?PRIME
    QOUT!PRIME %emit a discovered prime
    par
       FILTER(PRIME,QIN,Q)
       SIFT(Q,QOUT);

main
```

```
chan Q1, Q2
   par
     INTEGERS(Q1)
     SIFT(Q1,Q2)
     OUTPUT(Q2)
```