```
let factnk = rec(\f.\n.\k.
    if (isZero?(k),
        1,
        f(n - 1)(k - 1))) in

dest = {customer, true/false}
val = {n, k}
helper = \dest.\val send(first(dest), pair(factnk(first(val), second(val)), second(dest)))


num1 = {value, true/false -> if true then its the numerator)
num2 = same thing
join = rec(\dest.\num1.ready(\num2.
    if (second(num1),
        send(dest, first(num1) / first(num2)),
        send(dest, first(num2) / first(num1))

b = recursive function
m = {customer, {n, k}}
comb = (\b.\m. let
    cust = first(m)
    n = first(second(m))
    k = second(second(m))
    joiner = new(join(cust))
    numerator = new(helper(pair(joiner, true)))
    denominator = new(helper(pair(joiner, false))) in
    seq(
        send(numerator, pair(n, k)),
        send(denominator, pair(k, k)),
        ready(b)
    )

erlang solution:
-module(combinations).
-export([start/2, comb/4, factnk/2]).

factnk(_, 0) -> 1;
factnk(N, K) -> factnk(N-1, K-1).

comb(N, K, Case, Customer) -> Customer ! {Case, factnk(N, K)}.

start(N, K) ->
    spawn(combinations, comb, [N, K, num, self()]),
    spawn(combinations, comb, [K, K, den, self()]),

    receive {num, Num} -> receive {den, Den} -> Num / Den end end.

Some syntax things to be aware of: use rem instead of mod (I rem K), /= is not equals
```