Problem: generate all prime numbers between 1 and 1000

We'll consider different ideas using concurrency and Linda.

Text book uses eval (not implemented in Python Linda)

Example: eval (3, "xm", f(2.5), g("a"))

1. First, start some threads to calculate f(2.5) and g("a")
   — these calculations could be code that uses Linda operations
2. After these threads finish the computations — giving values for f(2.5) and g("a") — add the result
   (3, "xm", f(2.5), g("a"))
   to the tuple space

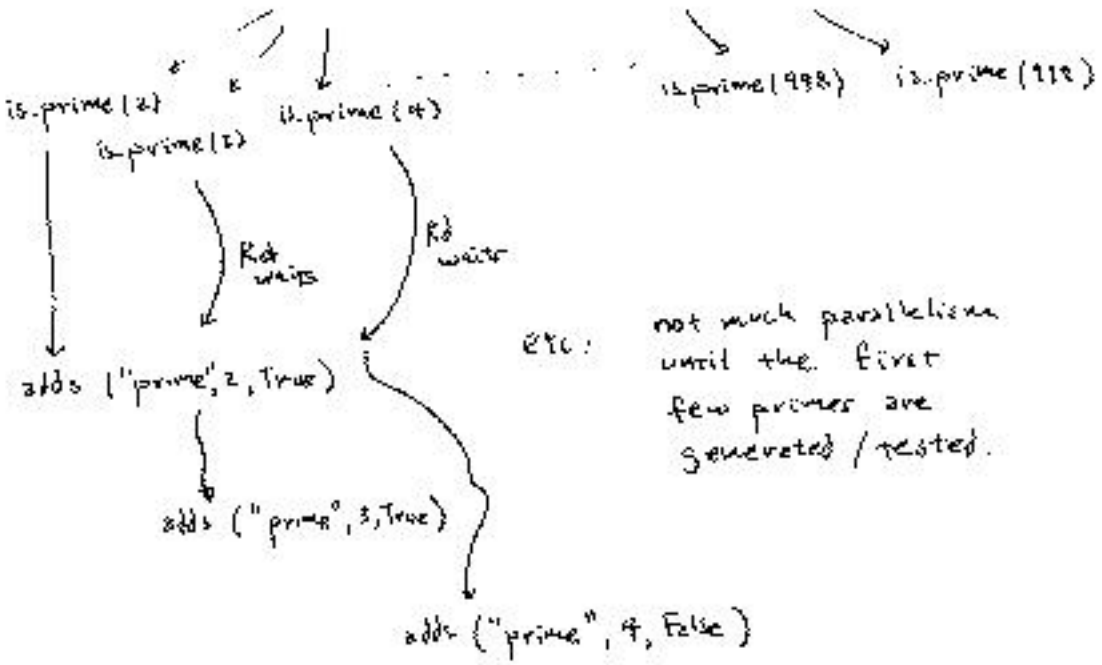
Program to generate primes

```
for i in range(2, 1000):
    eval("primes", i, is_prime(i))
for i in range(2, 1000):
    b = Rd(("primes", i, None))
    if b[2]: print i
```

```
import math
def is_prime(x):
    limit = math.sqrt(float(x))
    limit = int(math.ceil(limit))
    limit += 1
    if limit >= x: limit = x
    for j in range(2, limit):
        t = Linda.Rd(("primes", j, None))
        if t[2] and x % t == 0: return False
    return True
```

How can this be parallel?

- program generates "eval" -threads



Second Idea:

find primes in batches

The "granularity" is the number of candidate numbers in a batch (called a "grain")

We'll use master-worker paradigm

Master: to generate agenda of worker

```
for i in range(2, 1000, grain):
    Out(("task", i))
```

OR, alternatively the workers can generate the agenda

Master: `Out(("task", 2))`

Worker:

```
t = In(("task", None))
if t[1] + grain < 1000:
    Out(("task", t[1] + grain))
```

How does Master collect the results?

Workers can build result tuples:

`("result", k, s)`

- k: starting number of a batch
- s: string containing True/False results for the batch

Master:

```
for i in range(2, 1000, grain):
    v = In(("result", i, None))
    then get results from v[2]
```