xv6 Pipe-Based Concurrent Prime Sieve - Design Notes & Diagrams

1) Detailed Flow with Tiny Example (2…15) — print-as-discovered

```
Gen ──▶ 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Stage(2):
  read first = 2 → print "prime 2"
  forward: 3 5 7 9 11 13 15  (drop multiples of 2)

Stage(3):
  read first = 3 → print "prime 3"
  forward: 5 7 11 13         (drop multiples of 3)

Stage(5):
  read first = 5 → print "prime 5"
  forward: 7 11 13           (drop multiples of 5)

Stage(7):
  read first = 7 → print "prime 7"
  forward: 11 13

Stage(11):
  read first = 11 → print "prime 11"
  forward: 13

Stage(13):
  read first = 13 → print "prime 13"
  forward: (empty)

EOF cascades; stages exit in reverse order.
```

2) Stage Creation & FD Ownership (Process indices start at 0)

Mapping (indices → roles):

- **P0**: Generator (writes 2..280)
- **P1**: Stage[1] — discovers prime 2
- **P2**: Stage[2] — discovers prime 3
- **P3**: Stage[3] — discovers prime 5
- … and so on

2.1 Generic stage creation (from Pi to P(i+1))

Before `fork()` (in Pi):

```
owns:  Li.r          Ri.r  Ri.w
       (left/read)    (new)  (new)
```

After `fork()`:

```
Parent  Pi (filter for p_i):         Child  P(i+1) (next stage):
  keep:  Li.r, Ri.w                     keep:  Ri.r
  close: Ri.r                           close: Li.r, Ri.w
  do:    read Li.r; if x%p_i!=0 → Ri.w  do:    primes(Ri.r)  // tail
call
```

On upstream EOF in Pi:

```
close(Li.r); close(Ri.w)  → EOF to P(i+1)
wait(0); exit(0)
```

Where `Li` is the pipe **feeding** Stage[i] (for i=1, `L1 = IN`), and `Ri` is the pipe Stage[i] **creates** for Stage[i+1].

2.2 First two transitions

P0 → P1 (Stage[1], p1=2):

```
P0 creates IN; fork
  P0: writes 2..280 to IN.w, then close(IN.w)
  P1: primes(IN.r)  // Stage[1]
      read first=2 → print 2
      build R1; fork
        P1(parent): filter IN.r → R1.w (drop %2==0), then
close(IN.r,R1.w), wait, exit
        P2(child) : primes(R1.r) // Stage[2]
```

P1 → P2 (Stage[2], p2=3):

```
P2 reads first=3 → print 3
build R2; fork
  P2(parent): filter R1.r → R2.w (drop %3==0), then close(R1.r,R2.w),
wait, exit
  P3(child) : primes(R2.r) // Stage[3]
```

2.3 EOF cascade (left → right)

```
P0 closes IN.w  ➔  P1 sees EOF on L1.r
P1 closes R1.w  ➔  P2 sees EOF on L2.r (= R1.r)
P2 closes R2.w  ➔  P3 sees EOF on L3.r (= R2.r)
...
```

Each stage’s **parent** (the filter) closes its right write-end; once *all* writers for a pipe are closed, the next stage’s `read()` returns 0 (EOF), and termination unwinds cleanly.

---

3) FD Ownership Checklists

Parent of Stage[i] (filter):

- Close `Ri.r` immediately (you only write to the right pipe).
- Keep `Li.r` (read) and `Ri.w` (write) during filtering.
- After the loop: `close`(Li.r); `close`(Ri.w); `wait(0)`; `exit(0)`.

Child (next stage):

- Close `Ri.w` and `Li.r` immediately.
- Tail-call `primes`(Ri.r) (never returns).
