Concurrency patterns in interactive monitors

Raphael 'kena' Poss
University of Amsterdam
29-05-2013

Application domain

- Theme: "system monitoring and coordination"
- Setting: 2 systems
  - subject: system being controlled
  - controller: system doing the control
- Example: data vs control planes in telcos
- Domain: **concurrency in the controller**

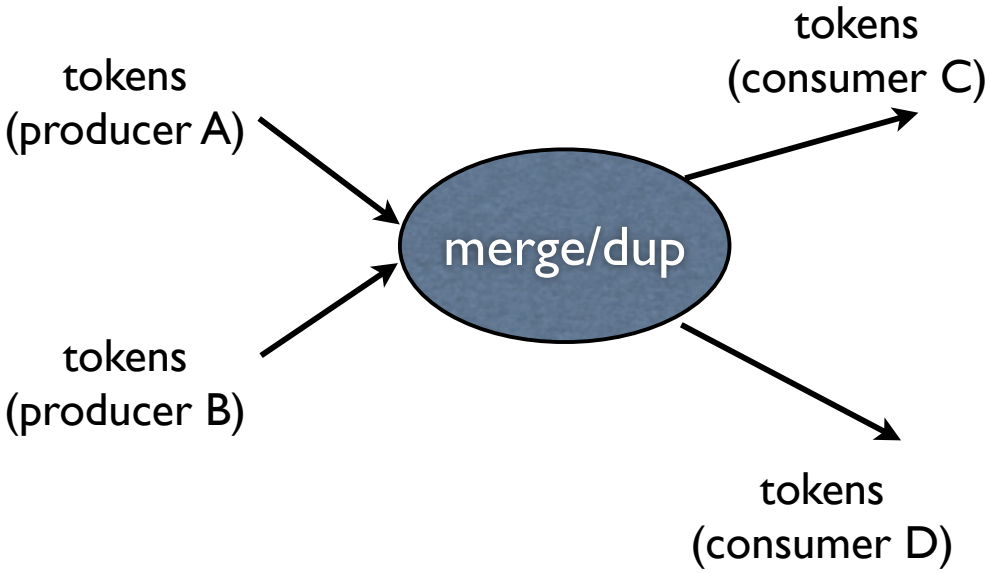
Key concept

- **2 independent discrete time domains**
  - control/reporting events with the "user" of the control system (either human or automated system)
  - sensor/actuator events with the subject system

Getting started

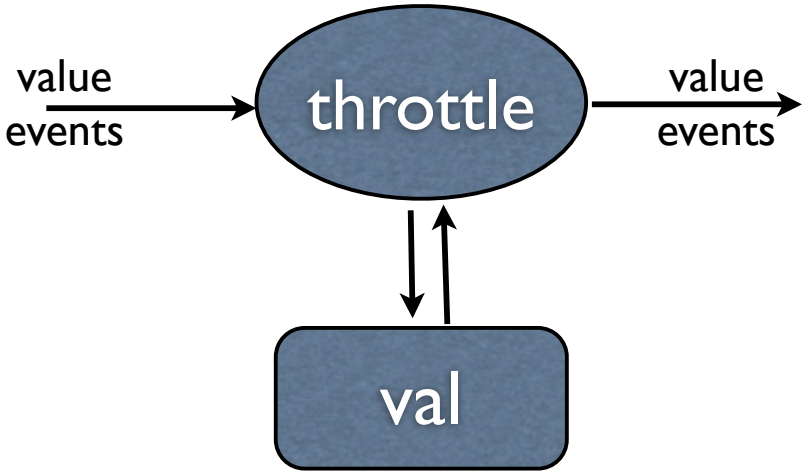
- The following slides show isolated patterns
- Pseudo-code is expressed in a CSP-like language:
  - variables can be scalars or channels
  - x := <- c means read x from channel c
  - c <- x means write x to channel c
  - select { cond1: ... cond2: ... } waits for either cond1 or cond2 to become possible, then performs the action associated with exactly 1 condition

Pattern: merge/dup



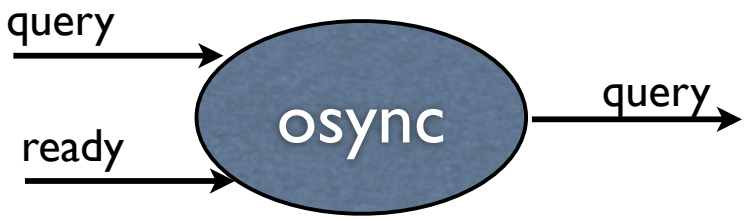

```
mergedup(src1 in, src2 in,
    prod1 out, prod2 out)
{
  repeat {
    select {
      v := <- src1
      v := <- src2
    }
    prod1 <- v
    prod2 <- v
  }
}
```

Pattern: throttle




```
throttle(per scalar,
         src in, prod out)
{
   val := 0
   repeat {
      v := <- src
      val += v
      if val >= per {
         prod <- val
         val := 0
      }
   }
}
```

Pattern: output sync



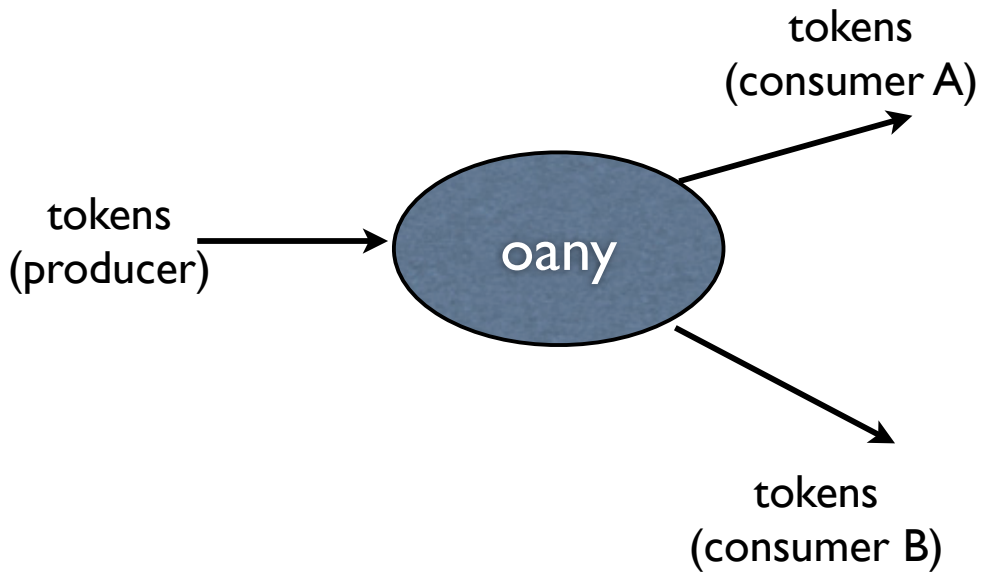
Contract:

1) input query events are fused until ready event received

2) each output query event consumes 1 ready event + 1 input query event

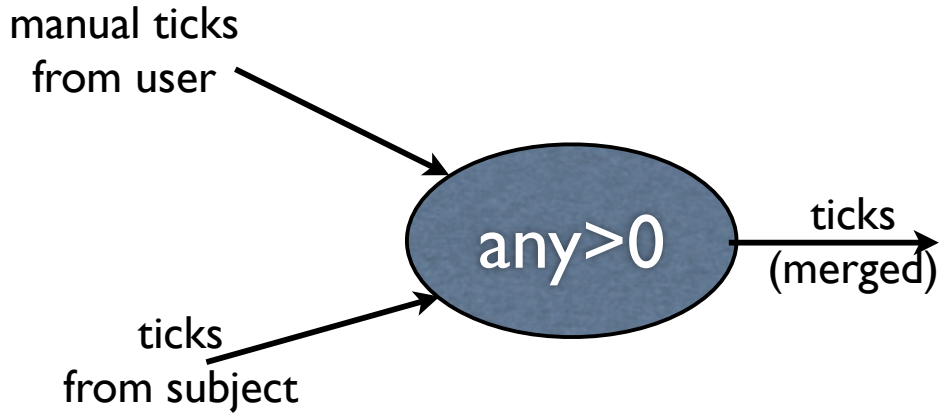
```
osync(qin in, qout out,
      rdy in)
{
  doit := false
  repeat {
     select {
        r := <- rdy: (nothing)
        q := <- qin:
             doit := true
             continue
     }
     if not doit {
          q := <- qin
     }

     qout <- q
  }
}
```

Pattern: output-any



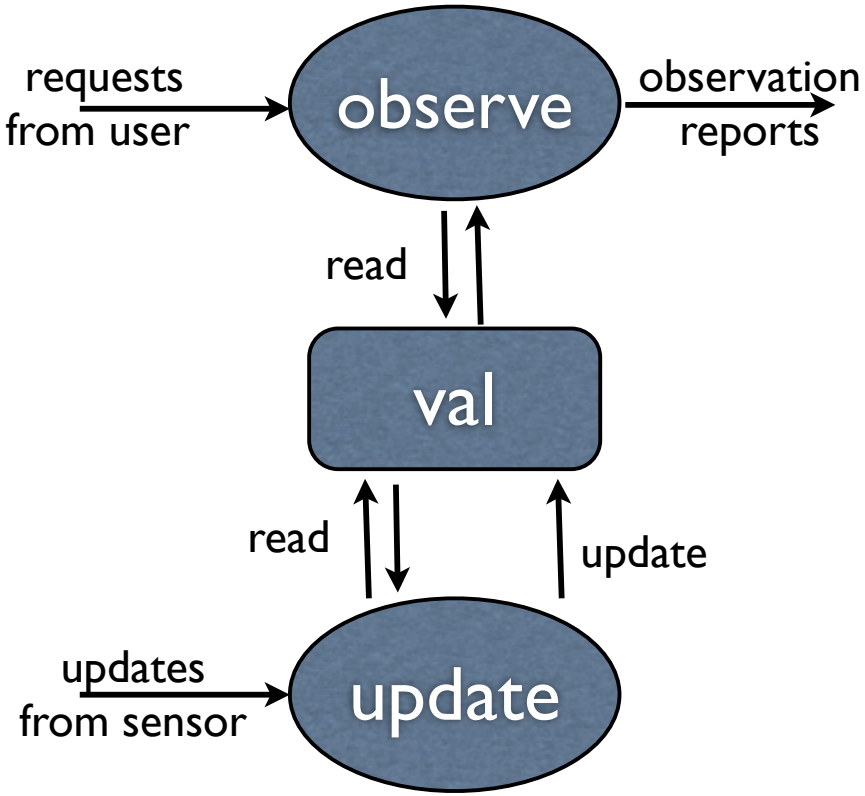

```
oany(src in, prod1 in,
     prod2 out)
{
   repeat {
       v := <- src
       select {
           prod1 <- v
           prod2 <- v
       }
    }
}
```

Pattern: any-filter



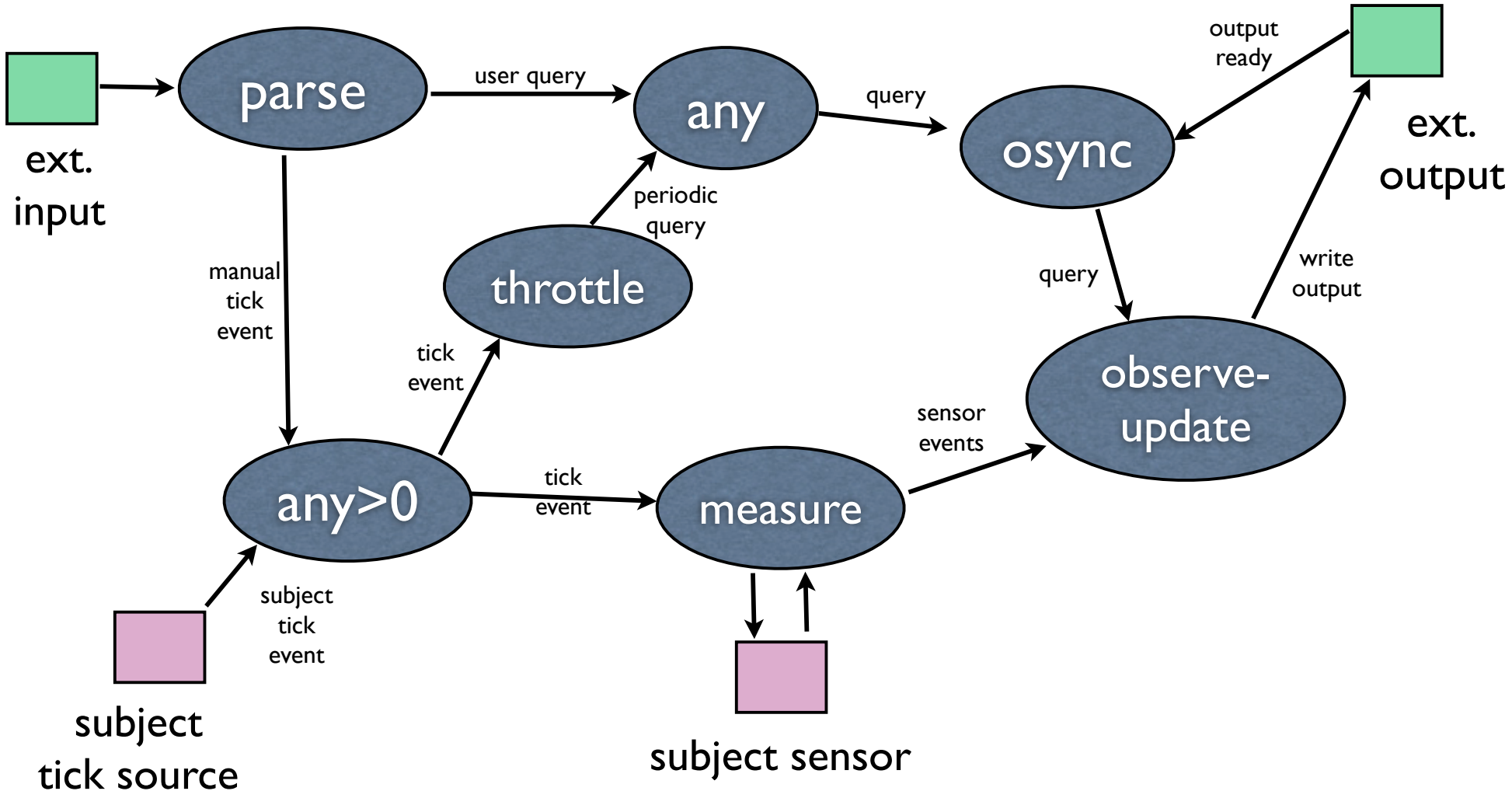

```
any(src1 in, src2 in,
    prod out)
{
  repeat {
    select {
      v := <- src1:
      v := <- src2:
    }
    if v > 0 {
      prod <- v
    }
  }
}
```

Pattern: sense-observe

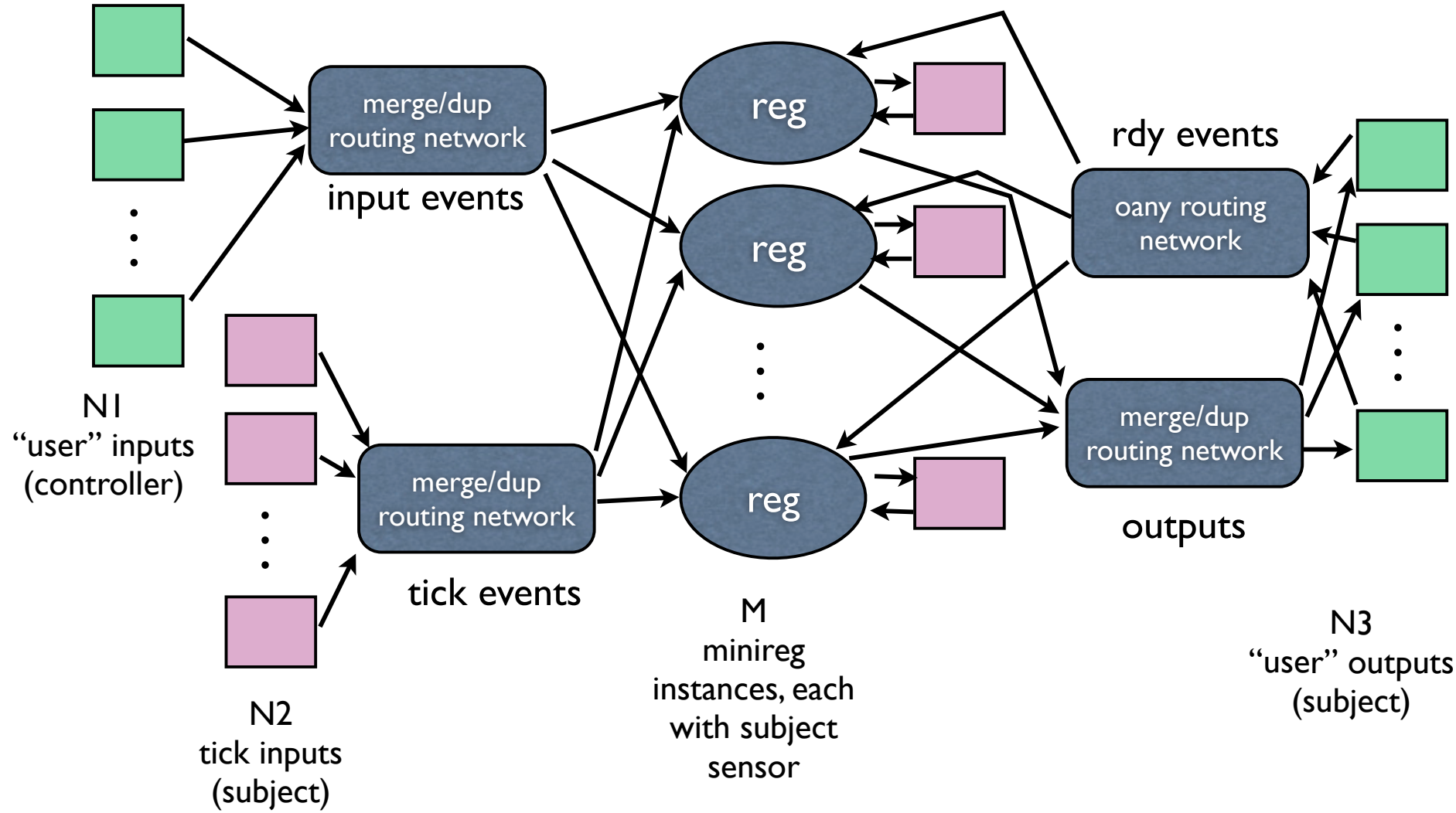



```
sense-observe(obsq in,
              obsr out,
              sense in)
{
   val := 0
   repeat {
     select {
        q := <- obsq:
           obsr <- val
        s := <- sense:
           val := f(val, s)
     }
   }
}
```

Example application: reg



Example application: multireg



Closing words

- I wrote an implementation of "reg" in Go: http://github.com/knz/reg
  - I'm looking for **any** language which makes this program simpler / shorter than 1kLOC
- Couldn't implement multireg in Go yet
  - Now looking for any suitable language
  - do you have suggestions?