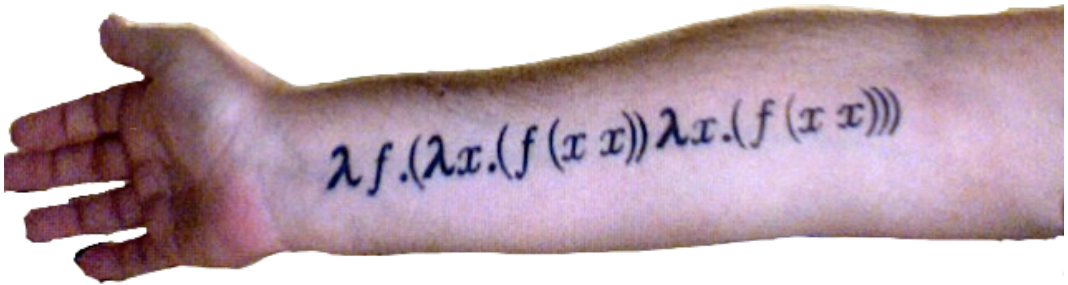
SSlang: A Sparse Synchronous Language for Hard Real-Time Tasks

Stephen A. Edwards et al.

CS @CU COMPUTER SCIENCE AT COLUMBIA UNIVERSITY

IFIP WG2.8 Functional Programming
May 2022, Cornell Tech, Roosevelt Island, NYC

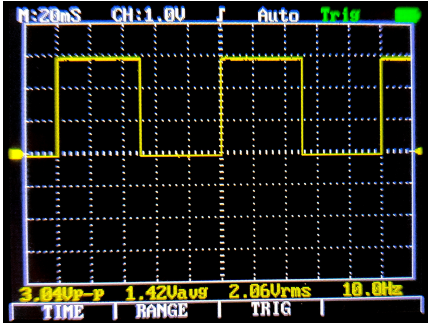

λf.(λx.(f (x x)) λx.(f (x x)))

```
gcd a b =                  // Inferred types
  if  a == b               // Indentation for grouping
    a                      // Everything is an expression
  else
    if  a < b              // User-defined binary infix operators
      gcd a (b − a)        // Juxtaposition for function calls
    else
      gcd (a − b) b
```

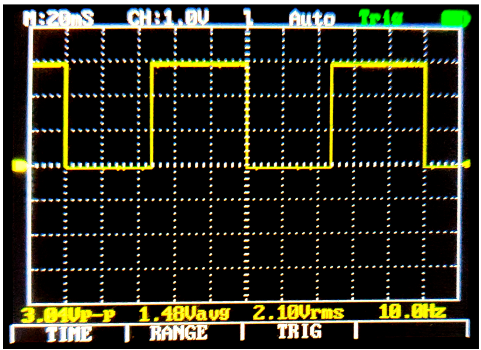
```
gcd a b c =                          // gcd : &Int → &Int → &Int → ()
  while deref a != deref b           // While loops
    if  deref a < deref b            // OCaml-like references
      b <− deref b − deref a         // Assignment to references
    else
      a <− deref a − deref b
  c <− deref a                       // Sequencing
```

```
add2 a = a <- deref a + 2
mult4 a = a <- deref a * 4

main =
  let a = new 1      // Allocate and name a new variable
  par add2 a         // Parallel function calls
      mult4 a        //   execute in prescribed order
                     // a is (1 + 2) * 4 = 12 here
  a <- 1
  par mult4 a
      add2 a
                     // a is (1 * 4) + 2 = 6 here
```

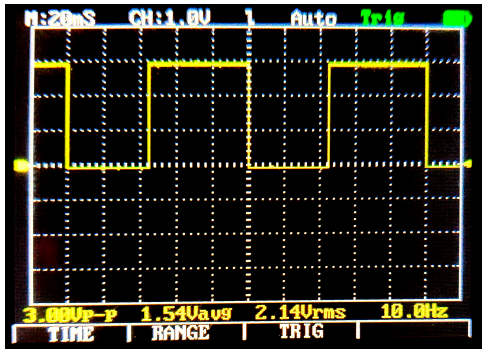
```
blink led =
  loop                          // Infinite loop
    after ms 50, led <- 1       // Schedule future variable update
    wait led                    // Block on variable update
    after ms 50, led <- 0
    wait led
```



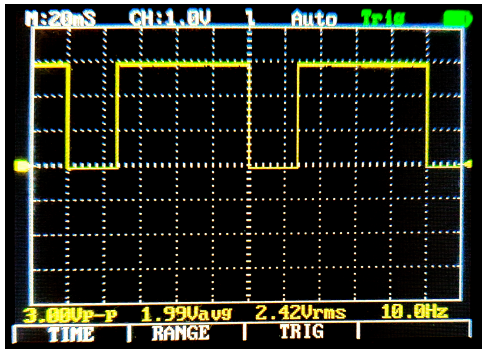
```
blink led =
  while 1

    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```

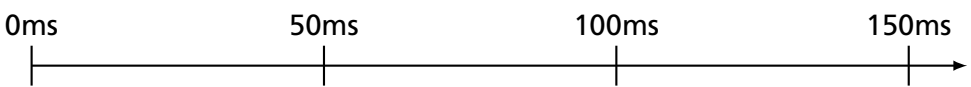


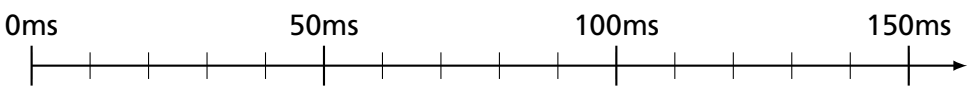
```
blink led =
  while 1
    fib 19
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```

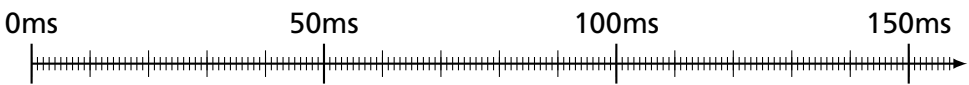


```
blink led =
  while 1
    fib 23
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```

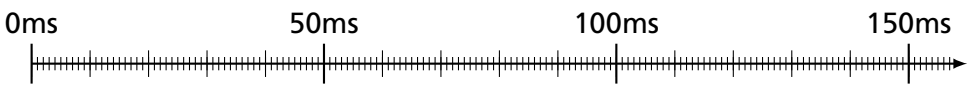



0ms
50ms
100ms
150ms

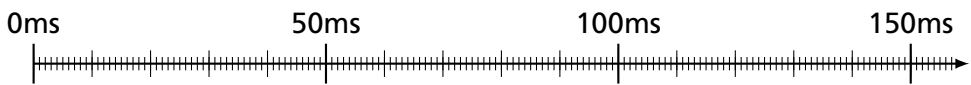

0ms
50ms
100ms
150ms


0ms
50ms
100ms
150ms

```
blink led =
   loop
      after ms 50, led <- 1
      wait led
      after ms 50, led <- 0
      wait led
```

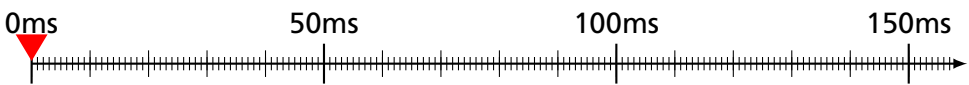


```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



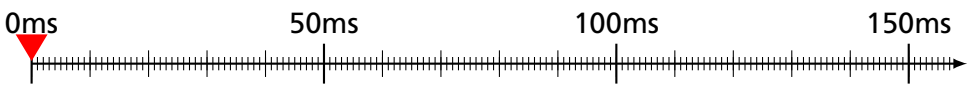
led 0

```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



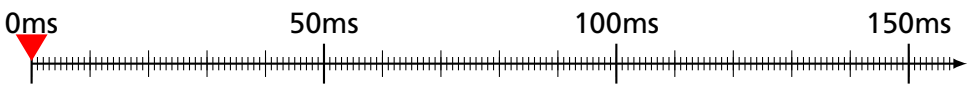
led 0

```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



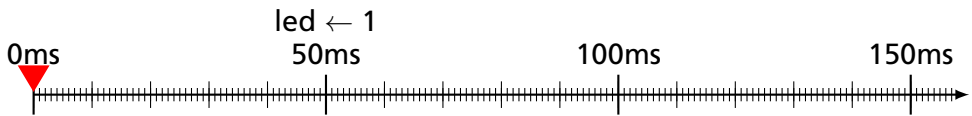
led 0

```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



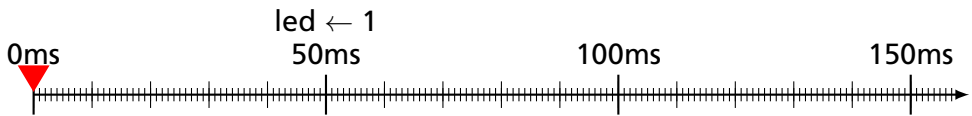
led 0

```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



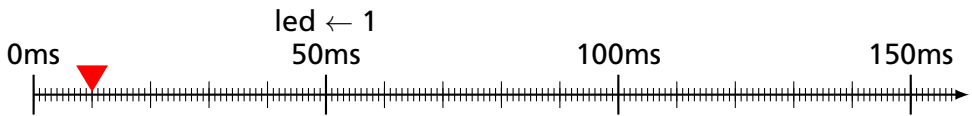
led 0

```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



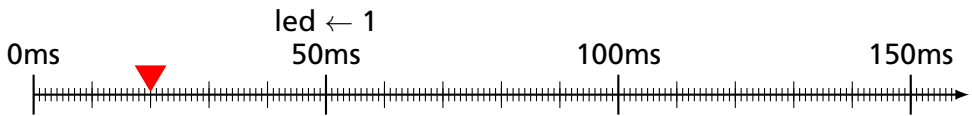
led 0

```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```

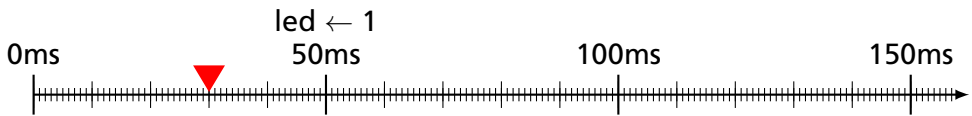


led

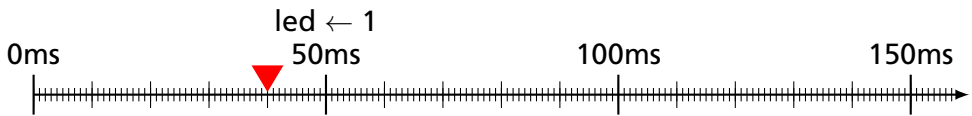
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



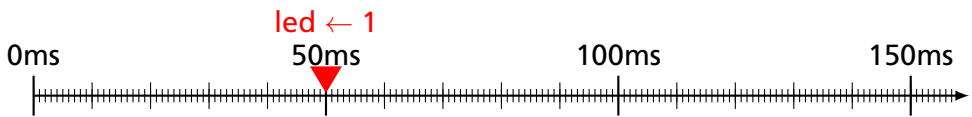
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



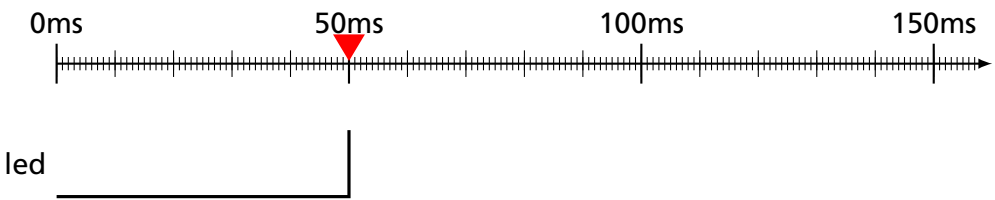
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



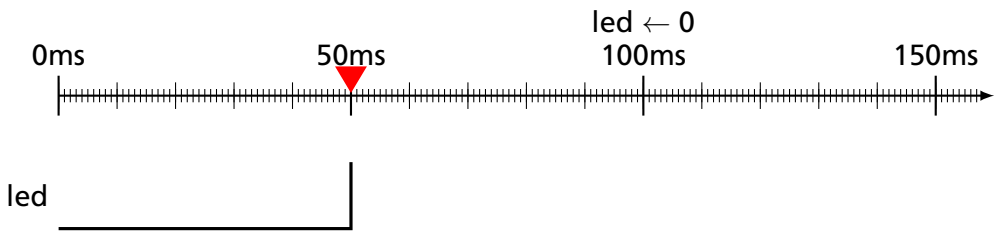
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



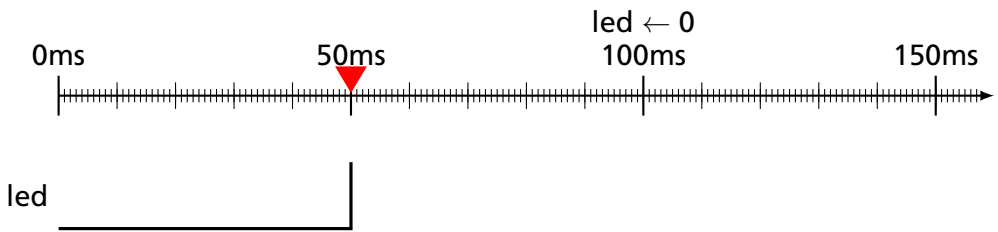
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



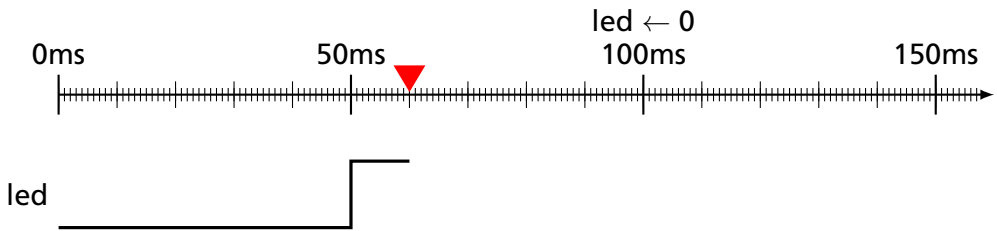
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



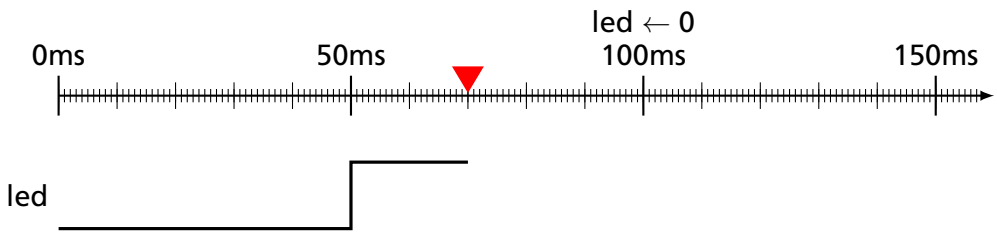
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



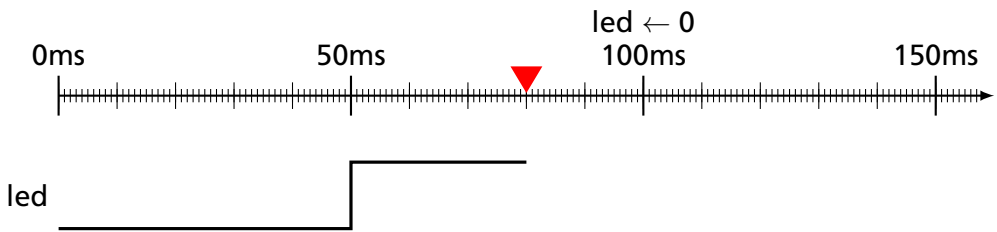
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



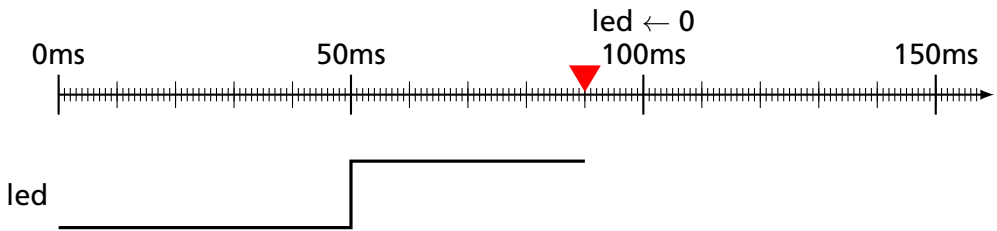
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



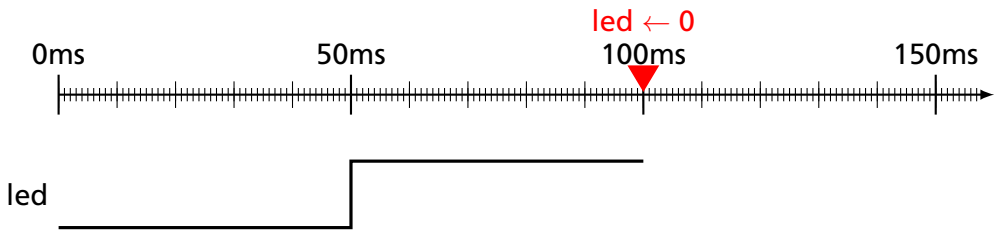
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



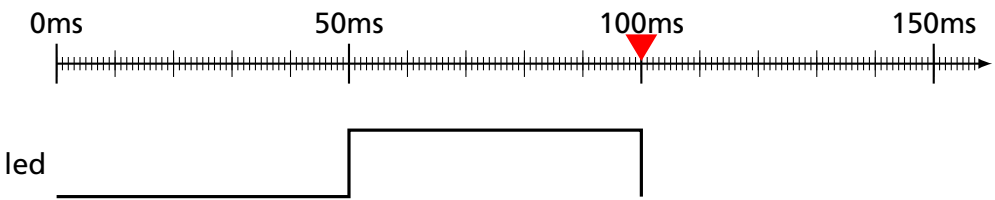
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



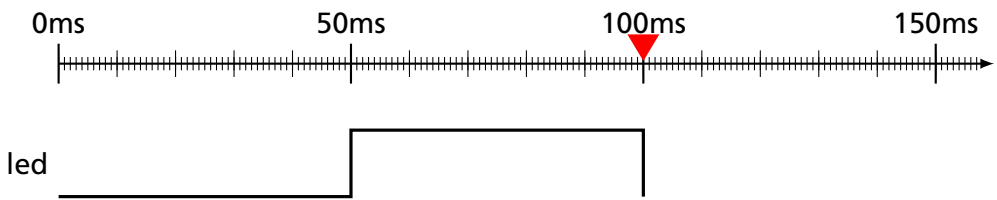
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



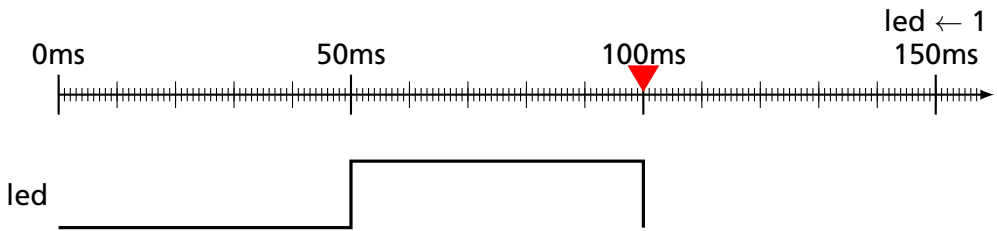
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



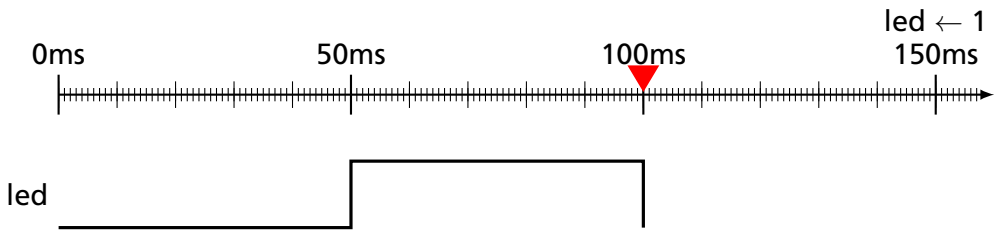
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



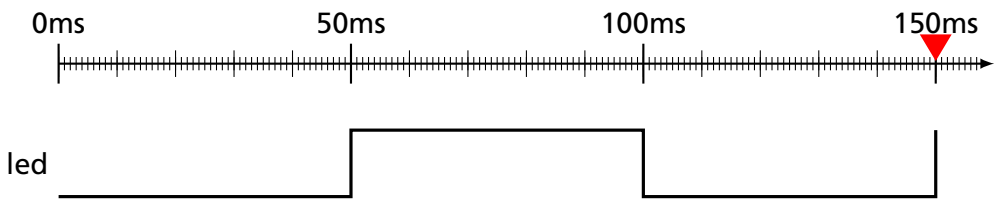
```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



```
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```



```
blink led period =
  let event = new ()        // Unit-valued variables pure events
  loop
    led <- 1 - deref led
    after period, event <- () // Schedule pure event
    wait event                // Wait on write, not change

main led =
  par blink led (ms 50)
      blink led (ms 30)
      blink led (ms 20)       // LED may toggle three times
```

- Deterministic concurrency
- Immutable and mutable values
- Algebraic data types, pattern matching
- Compiles to C for portability across microcontrollers
- Heap-resident function activation records
- Reference-counted heap, inspired by Perceus [PLDI 2021]
- No true parallelism (for now)
- No gradual typing (sorry)

Priority queue of events (time, variable, value), ordered by time

Priority queue of threads, ordered by priority

tick()
  While there are queued events now,
    Dequeue event $e = (\text{now}, v, n)$
    Update variable v with new value n
    Schedule each thread blocked on variable v
  While there are ready threads,
    Dequeue the lowest-priority thread t
    Run thread t from where it last blocked,
      which may write variables immediately to trigger threads now,
      or may schedule future variable update events

One event per variable: scheduling an update deletes any outstanding

Only “later”-priority threads are scheduled when a thread writes to a variable.

SSlang vs. Esterel

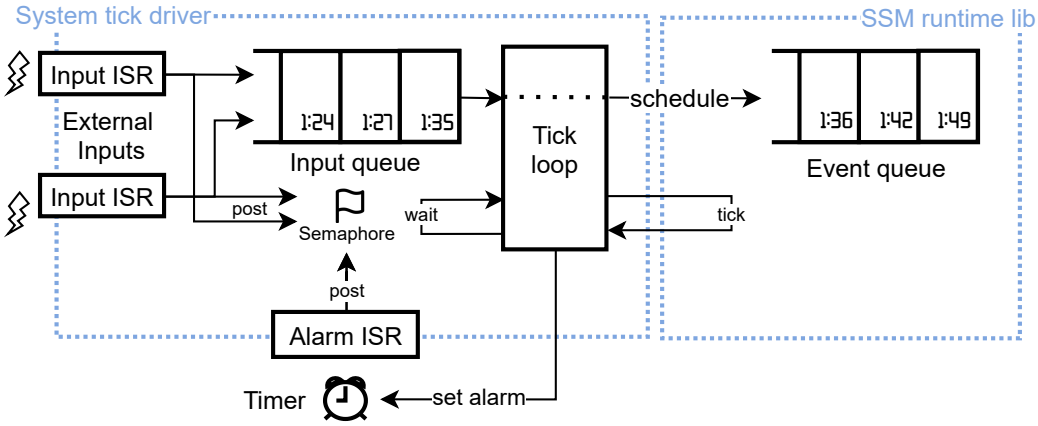
[Berry and Gonthier, SCP 1992]

	SSLang	**Esterel**
Deterministic	Yes	Yes
Time	Sparse	Dense
Within instants	Totally-ordered	Constructive
Runtime	Dynamic Event Queues	Statically Scheduled
Topology	Dynamic, recursive	Static

SSlang vs. Ptides

[Zhao, Liu, and Lee, RTAS 2007] [Zou Ph.D 2011]

	SSlang	**Ptides**
Between instants	Discrete-event	Discrete-Event
Within instants	Totally-ordered	Discrete-Event
Topology	Dynamic, recursive	Static
Implementation	Single-threaded	Distributed


System tick driver
SSM runtime lib
Input ISR
External
Inputs
Input ISR
post
1:24
1:27
1:35
Input queue
Semaphore
wait
Tick
loop
schedule
1:36
1:42
1:49
Event queue
tick
post
Alarm ISR
Timer
set alarm

https://github.com/ssm-lang/sslang