The Sparse Synchronous Model

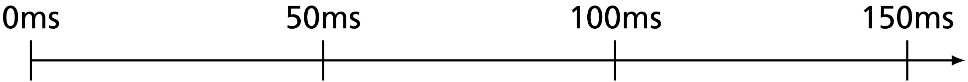
Stephen A. Edwards

COMPUTER SCIENCE AT
COLUMBIA UNIVERSITY

Chalmers, February 2, 2021

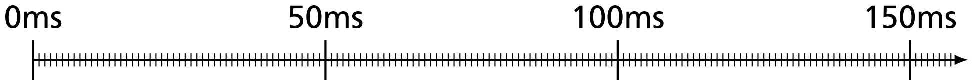
See also Edwards and Hui, FDL 2020

Time modeled arithmetically



Time modeled arithmetically

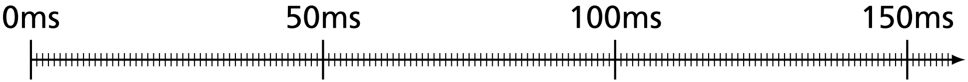
Quantized; quantum
not user-visible



Time modeled arithmetically

Quantized; quantum
not user-visible

Infinitely fast processor model:
Program execution a series of
zero-time instants
(hence “synchronous”)



Time modeled arithmetically

Quantized; quantum
not user-visible

Infinitely fast processor model:
Program execution a series of
zero-time instants
(hence “synchronous”)

Nothing happens in
most instants (hence “sparse”)

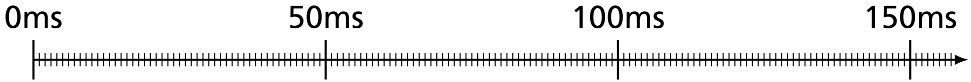
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <– 1
    wait led
    50 ms later led <– 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



led 0

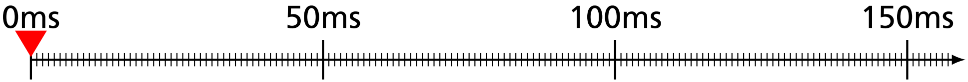
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



led 0

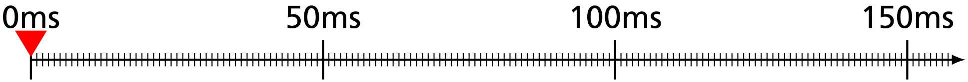
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <- 1
    wait led
    50 ms later led <- 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



led 0

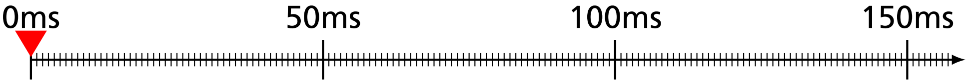
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <- 1
    wait led
    50 ms later led <- 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



led 0

```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable

led 0

```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable

led 0

```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable

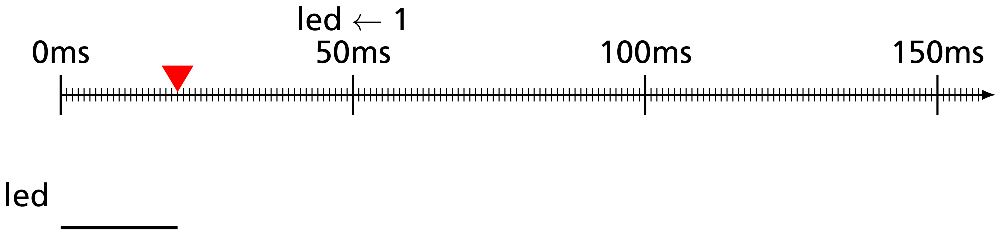
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <- 1
    wait led
    50 ms later led <- 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <- 1
    wait led
    50 ms later led <- 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

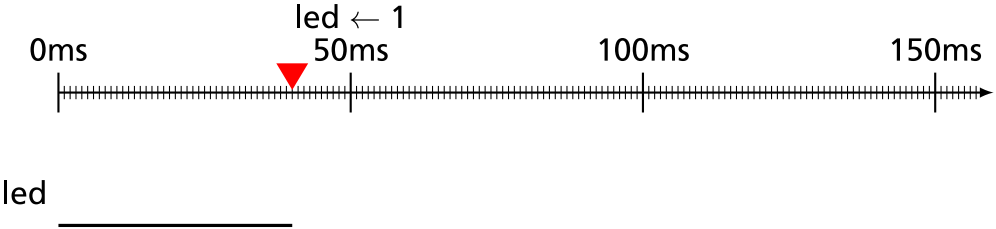
Infinite loop

Schedule a future update

Wait for a write on a variable

```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <– 1
    wait led
    50 ms later led <– 0
    wait led
```

- *led* is a pass-by-reference integer that can be scheduled
- Infinite loop
- Schedule a future update
- Wait for a write on a variable



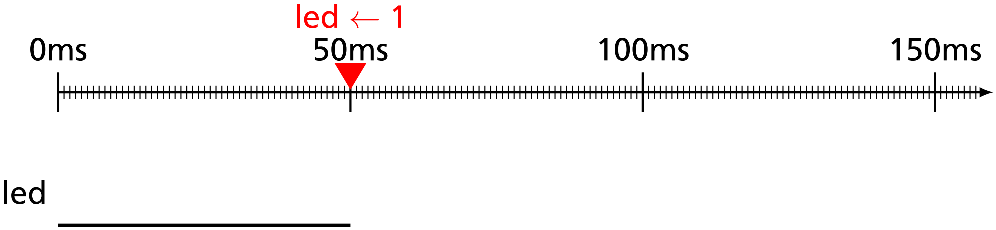
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



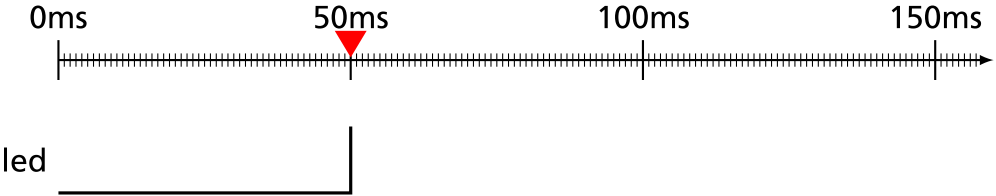
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <– 1
    wait led
    50 ms later led <– 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



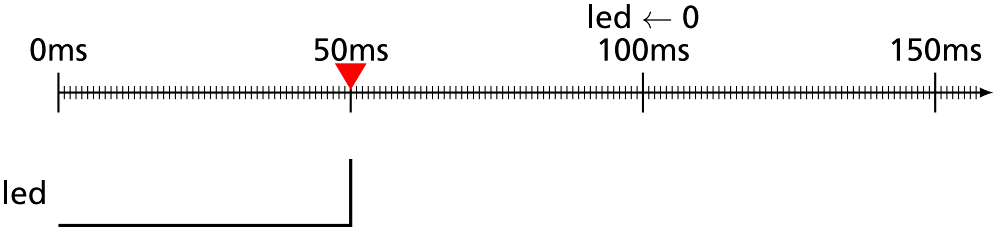
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



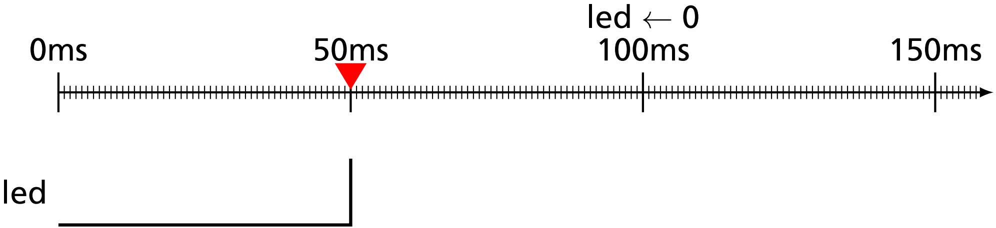
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <- 1
    wait led
    50 ms later led <- 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



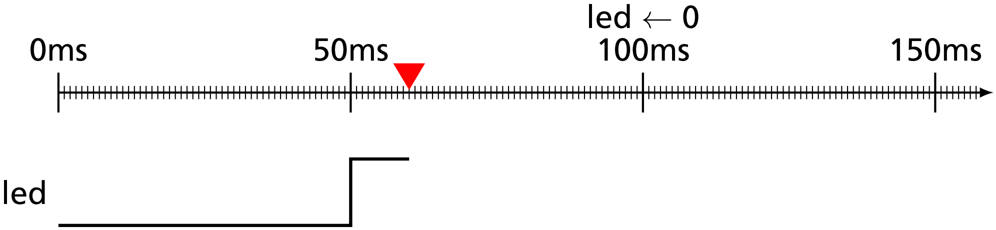
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <- 1
    wait led
    50 ms later led <- 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



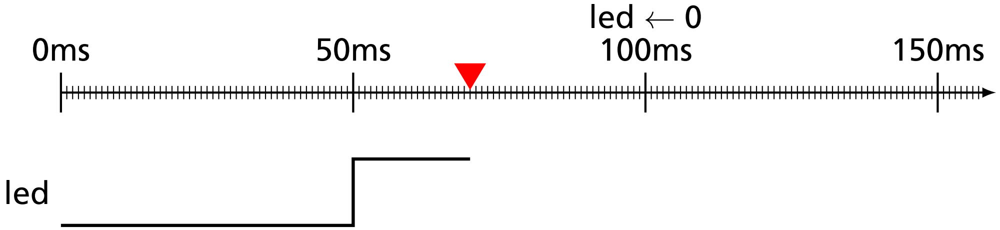
main(led : Ref (Sched Int)) =
  loop
    50 ms **later** led <— 1
    wait led
    50 ms **later** led <— 0
    wait led

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



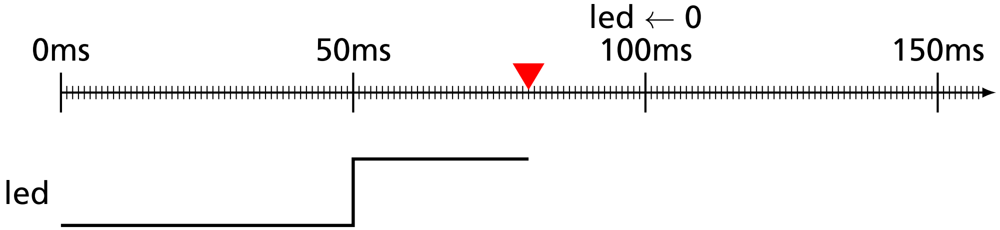
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



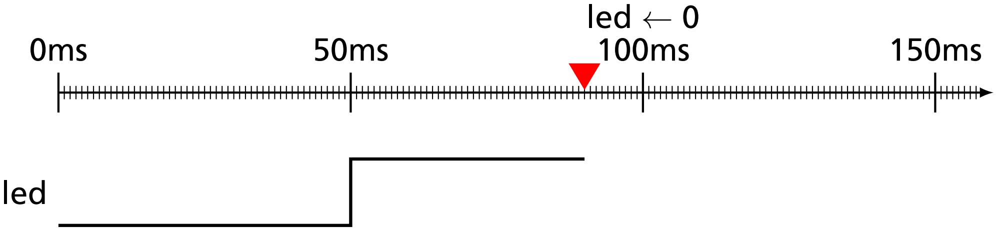
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <- 1
    wait led
    50 ms later led <- 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



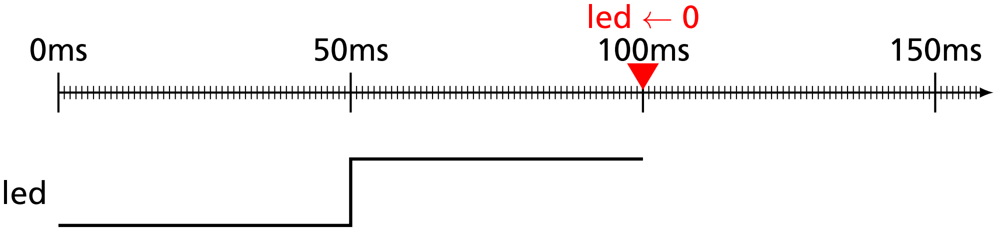
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



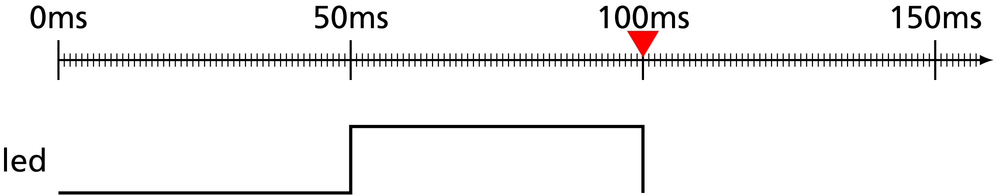
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <– 1
    wait led
    50 ms later led <– 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



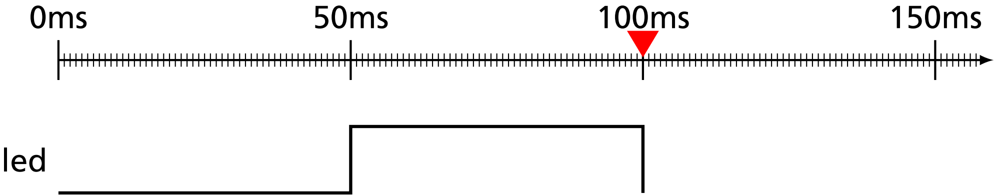
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <– 1
    wait led
    50 ms later led <– 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



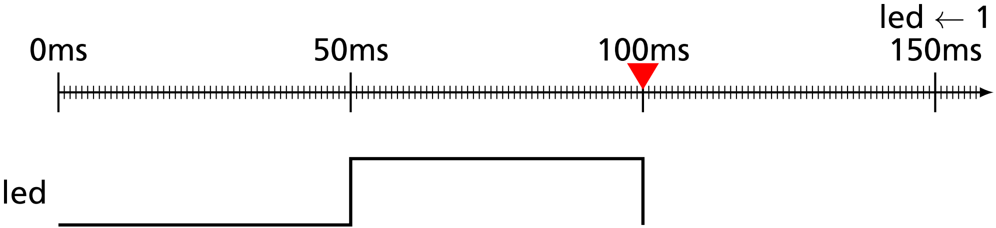
main(led : Ref (Sched Int)) =
  loop
    50 ms **later** led <— 1
    wait led
    50 ms **later** led <— 0
    wait led

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



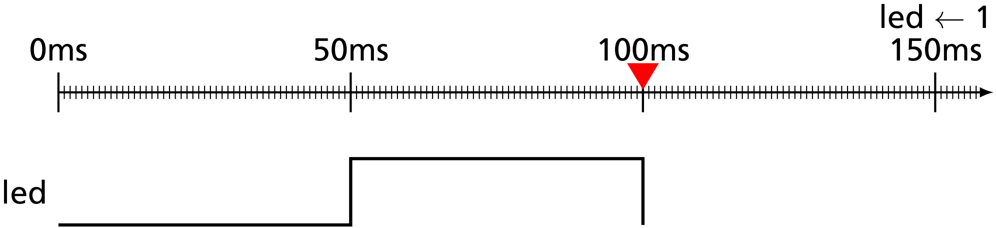
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

Wait for a write on a variable



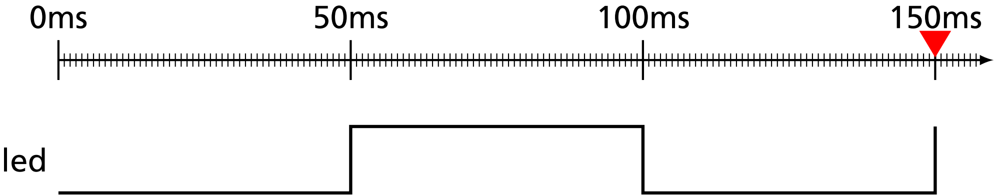
```
main(led : Ref (Sched Int)) =
  loop
    50 ms later led <- 1
    wait led
    50 ms later led <- 0
    wait led
```

led is a pass-by-reference integer that can be scheduled

Infinite loop

Schedule a future update

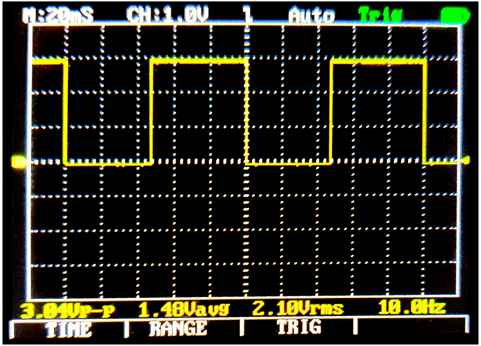
Wait for a write on a variable



Missing Deadlines Doesn't Affect Period

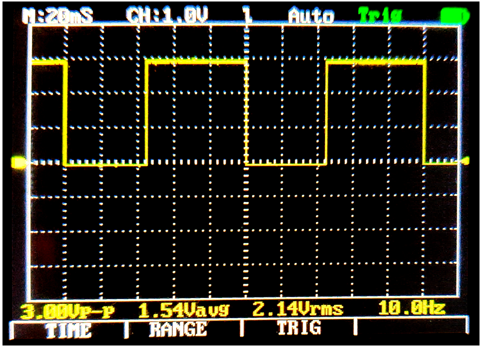
main(led : Ref (Sched Int)) =
loop

50 ms **later** led <— 1
wait led
50 ms **later** led <— 0
wait led



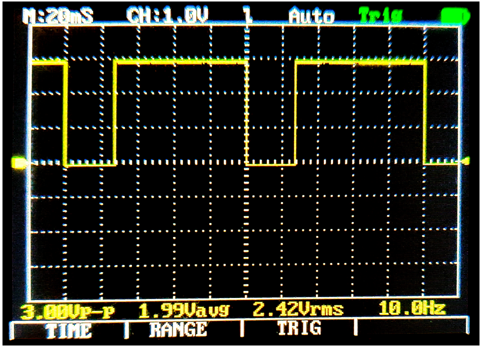
Missing Deadlines Doesn't Affect Period

main(led : Ref (Sched Int)) =
  loop
    fib 19 r
    50 ms **later** led <— 1
    wait led
    50 ms **later** led <— 0
    wait led



Missing Deadlines Doesn't Affect Period

```
main(led : Ref (Sched Int)) =
  loop
    fib 23 r
    50 ms later led <— 1
    wait led
    50 ms later led <— 0
    wait led
```



Recursive subroutines

```
toggle(led : Ref (Sched Int)) =
  led <− 1 − led
```

Pure events like "void" or "unit"

```
toggle(led : Ref (Sched Int)) =
  led <− 1 − led

slow(led : Ref (Sched Int)) =
  let e1 = Occur : Sched Event
```

Function call

```
toggle(led : Ref (Sched Int)) =
  led <– 1 – led

slow(led : Ref (Sched Int)) =
  let e1 = Occur : Sched Event
  loop
    toggle led
```

"Occur": only value of a pure event

```
toggle(led : Ref (Sched Int)) =
  led <– 1 – led

slow(led : Ref (Sched Int)) =
  let e1 = Occur : Sched Event
  loop
    toggle led
    30 ms later e1 <– Occur
    wait e1
```

Concurrent function calls

```
toggle(led : Ref (Sched Int)) =
  led <– 1 – led

slow(led : Ref (Sched Int)) =
  let e1 = Occur : Sched Event
  loop
    toggle led
    30 ms later e1 <– Occur
    wait e1
```

```
fast(led : Ref (Sched Int)) =
  let e2 = Occur : Sched Event
  loop
    toggle led
    20 ms later e2 <– Occur
    wait e2

main(led : Ref (Sched Int)) =
  pipe slow led
       fast led
```

Concurrent Routines Execute in Syntactic Order for Determinism

```
main()
  let a = 1 : Int
  pipe foo a
       bar a
```

Concurrent Routines Execute in Syntactic Order for Determinism

```
foo(a : Ref Int) =
  a <− a + 2

bar(a : Ref Int) =
  a <− a ∗ 4
```

```
main()
  let a = 1 : Int
  pipe foo a
       bar a
```

Concurrent Routines Execute in Syntactic Order for Determinism

```
foo(a : Ref Int) =
  a <- a + 2

bar(a : Ref Int) =
  a <- a * 4
```

```
main()
  let a = 1 : Int
  pipe foo a
       bar a
  // foo runs first: a = 12 = (1 + 2) * 4
```

Concurrent Routines Execute in Syntactic Order for Determinism

```
foo(a : Ref Int) =
  a <- a + 2

bar(a : Ref Int) =
  a <- a * 4
```

```
main()
  let a = 1 : Int
  pipe foo a
       bar a
  // foo runs first: a = 12 = (1 + 2) * 4
  pipe bar a
       foo a
```

Concurrent Routines Execute in Syntactic Order for Determinism

```
foo(a : Ref Int) =
  a <- a + 2

bar(a : Ref Int) =
  a <- a * 4
```

```
main()
  let a = 1 : Int
  pipe foo a
       bar a
  // foo runs first: a = 12 = (1 + 2) * 4
  pipe bar a
       foo a
  // bar runs first: a = 50 = (12 * 4) + 2
```

SSM vs. Esterel

[Berry and Gonthier, SCP 1992]

SSM vs. Esterel

[Berry and Gonthier, SCP 1992]

	SSM	**Esterel**
Deterministic	Yes	Yes

SSM vs. Esterel

[Berry and Gonthier, SCP 1992]

	SSM	**Esterel**
Deterministic	Yes	Yes
Time	Sparse	Dense

SSM vs. Esterel

[Berry and Gonthier, SCP 1992]

	SSM	**Esterel**
Deterministic	Yes	Yes
Time	Sparse	Dense
Within instants	Totally-ordered	Constructive

SSM vs. Esterel

[Berry and Gonthier, SCP 1992]

	SSM	**Esterel**
Deterministic	Yes	Yes
Time	Sparse	Dense
Within instants	Totally-ordered	Constructive
Compilation	Separate	Whole-program

SSM vs. Esterel

[Berry and Gonthier, SCP 1992]

	SSM	**Esterel**
Deterministic	Yes	Yes
Time	Sparse	Dense
Within instants	Totally-ordered	Constructive
Compilation	Separate	Whole-program
Runtime	Dynamic Event Queues	Statically Scheduled

SSM vs. Esterel

[Berry and Gonthier, SCP 1992]

	SSM	Esterel
Deterministic	Yes	Yes
Time	Sparse	Dense
Within instants	Totally-ordered	Constructive
Compilation	Separate	Whole-program
Runtime	Dynamic Event Queues	Statically Scheduled
Topology	Dynamic, recursive	Static

SSM vs. Ptides

[Zhao, Liu, and Lee, RTAS 2007]

SSM vs. Ptides

[Zhao, Liu, and Lee, RTAS 2007]

	SSM	Ptides
Between instants	Discrete-event	Discrete-Event

SSM vs. Ptides

[Zhao, Liu, and Lee, RTAS 2007]

	SSM	**Ptides**
Between instants	Discrete-event	Discrete-Event
Within instants	Totally-ordered	Discrete-Event

SSM vs. Ptides

[Zhao, Liu, and Lee, RTAS 2007]

	SSM	**Ptides**
Between instants	Discrete-event	Discrete-Event
Within instants	Totally-ordered	Discrete-Event
Topology	Dynamic, recursive	Static

SSM vs. Ptides

[Zhao, Liu, and Lee, RTAS 2007]

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

[Zou Ph.D 2011] See also Lee, Lohstroh et al. *Linga Franca*

Compared to Dynamic Ticks

Haxlenden, Bourke, Girault, FDL 2017

Dynamic ticks uses repeated “min” to decide “how long to wait”

SSM uses an event (priority) queue to decide this

Dynamic Ticks uses the richer, but harder-to-compile Esterel semantics

Compared to Boussinot's Work

Boussinot's schedule-based-on-syntactic-order inspired the SSM policy

Boussinot: Round-robin cooperative scheduler; SSM: totally-ordered-within-an-instant

Less concern for real-time behavior; more an operational replacement for Esterel-style semantics https://github.com/sedwards-lab/peng