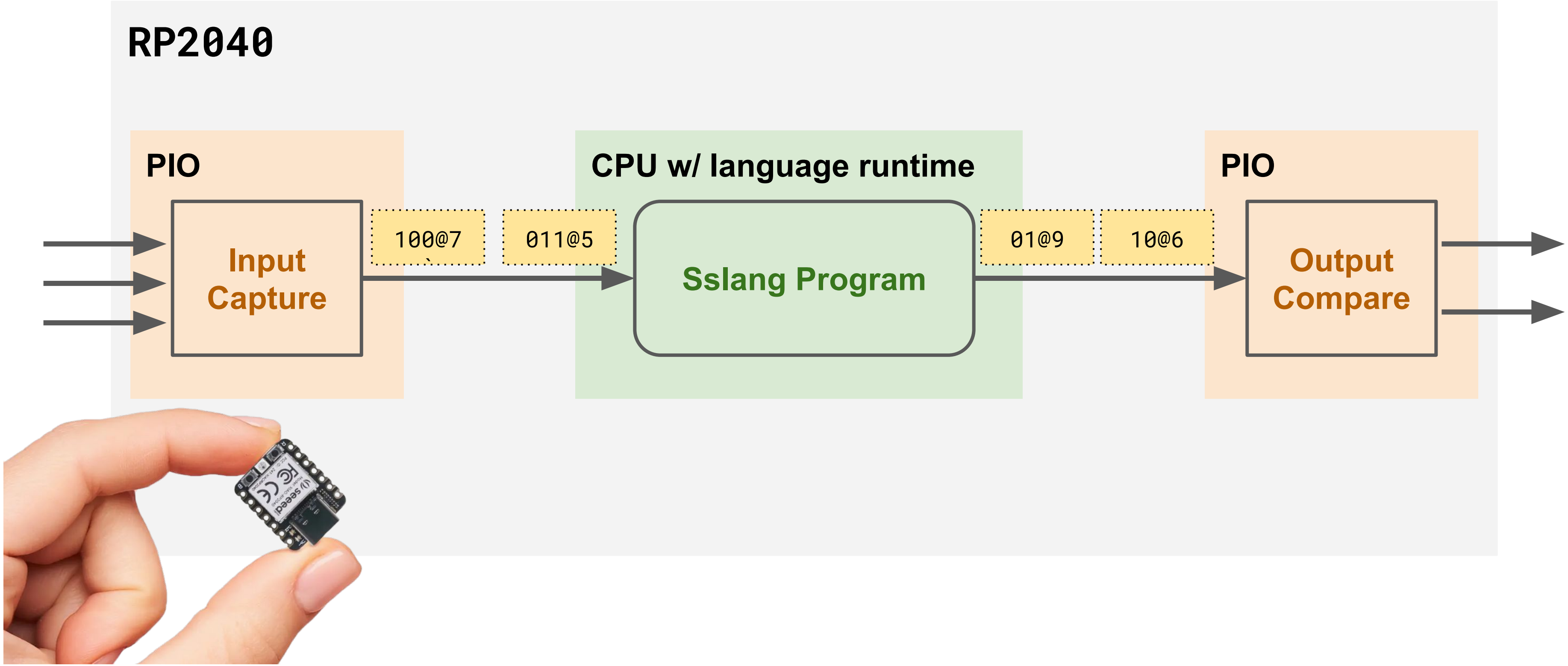
Timestamp Peripherals for Precise Real-Time Programming

John Hui Kyle J. Edwards Stephen A. Edwards

COMPUTER SCIENCE AT
COLUMBIA UNIVERSITY

MEMOCODE. Hamburg, Germany. September 22, 2023.



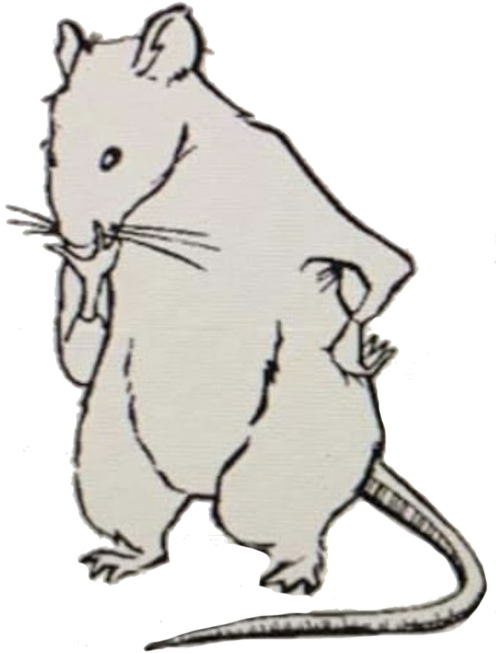
RP2040
PIO
Input Capture
100@7
011@5
CPU w/ language runtime
Sslang Program
01@9
10@6
PIO
Output Compare

What is the right programming model?

Programs should **compose**, but we have limited **hardware timers**

Polling input (in software) is wasteful: events are **bursty**

Timing **prescriptions** should not depend on device **clock rate**



Sslang at a glance

Procedural language with ML-like (**functional**) features:

polymorphism, static type inference, first-class functions, automatic memory management

```
waitfor (var: &a) (val: a) =   // type: ∀a, &a -> a -> ()
  while deref var != val       // current value is not val
    wait var                   // suspend until next update
```

Extended with ***synchronous*** primitives: **`after`**, **`wait`**, **`par`** (more on this later)

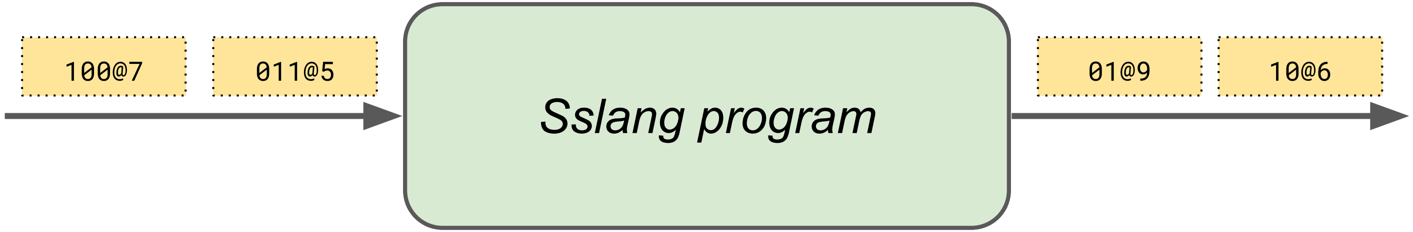
All computation other than **`wait`** takes zero logical time

Sslang at a glance

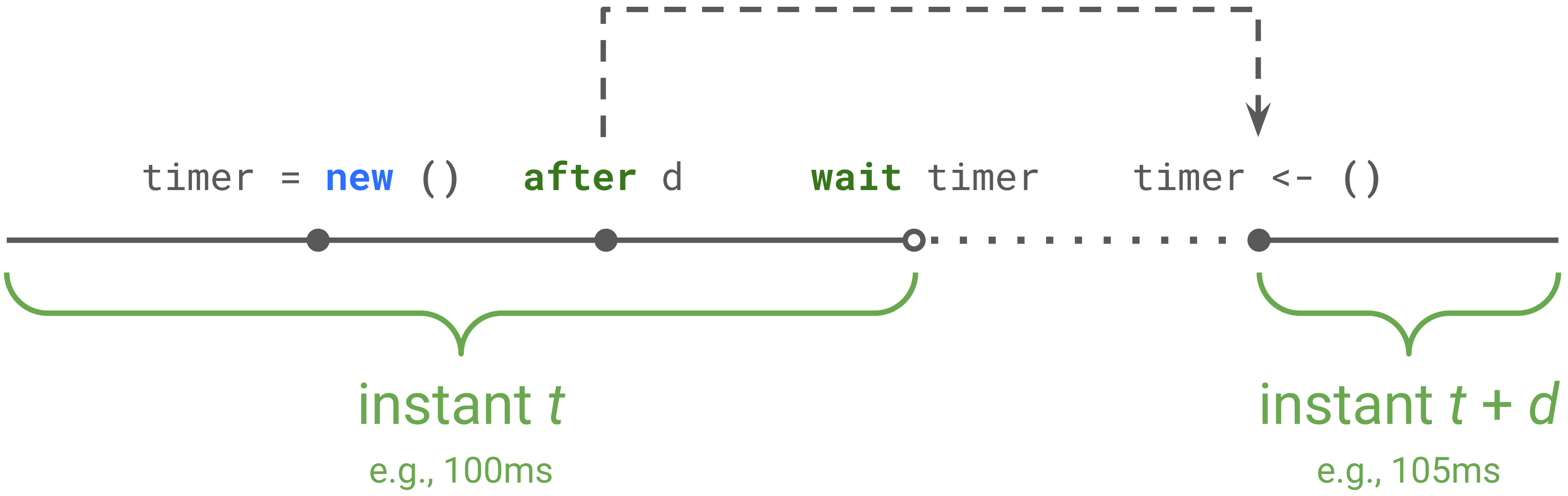
Variables (&) are mutable references

```
new     :  a -> &a           // construct variable from value
deref   :  &a -> a           // read current value of variable
_ <- _  :  &a -> a -> ()     // update value of variable
```

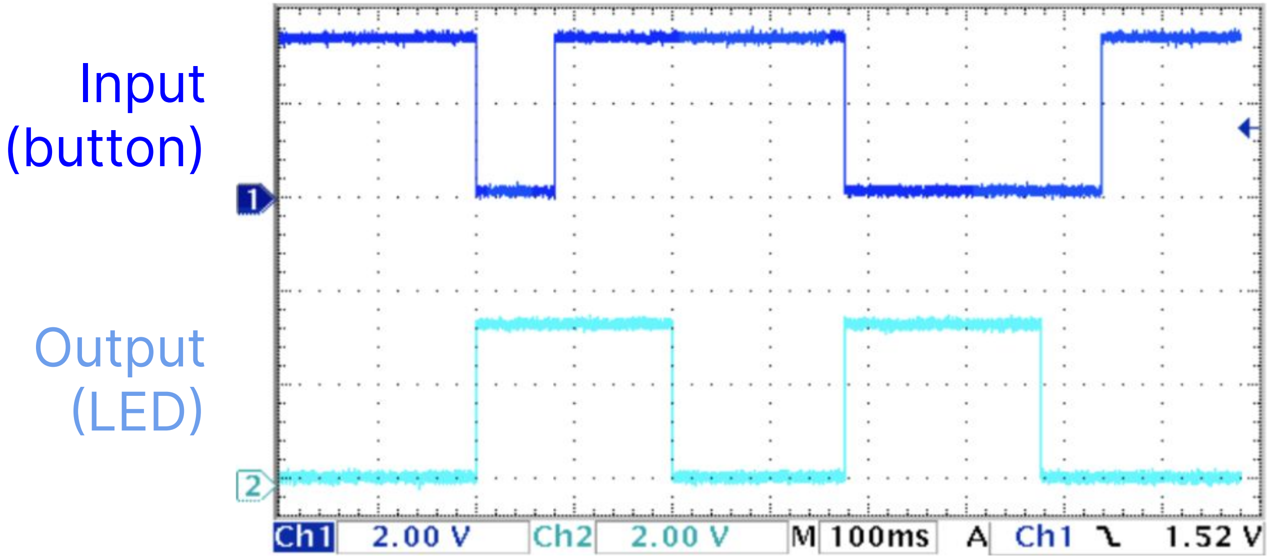
Variables convey buffered ***events*** (value + timestamp), both internal and external



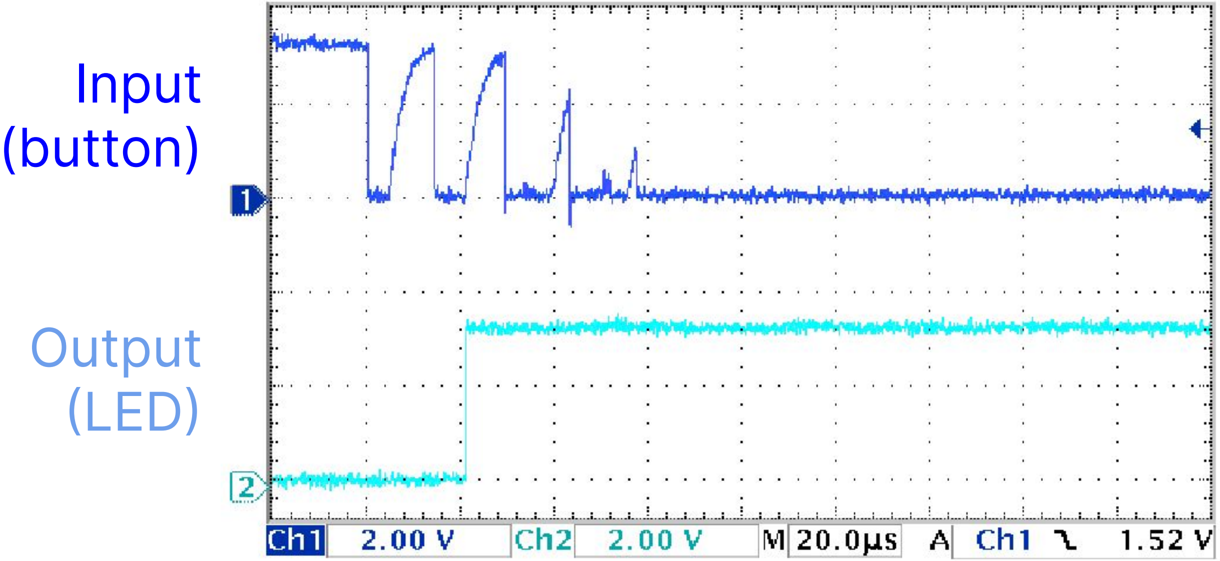
```
sleep (d: Time) =
  let timer = new ()      // construct event variable
  after d, timer <- ()    // schedule a wake-up event
  wait timer              // suspend until then
```



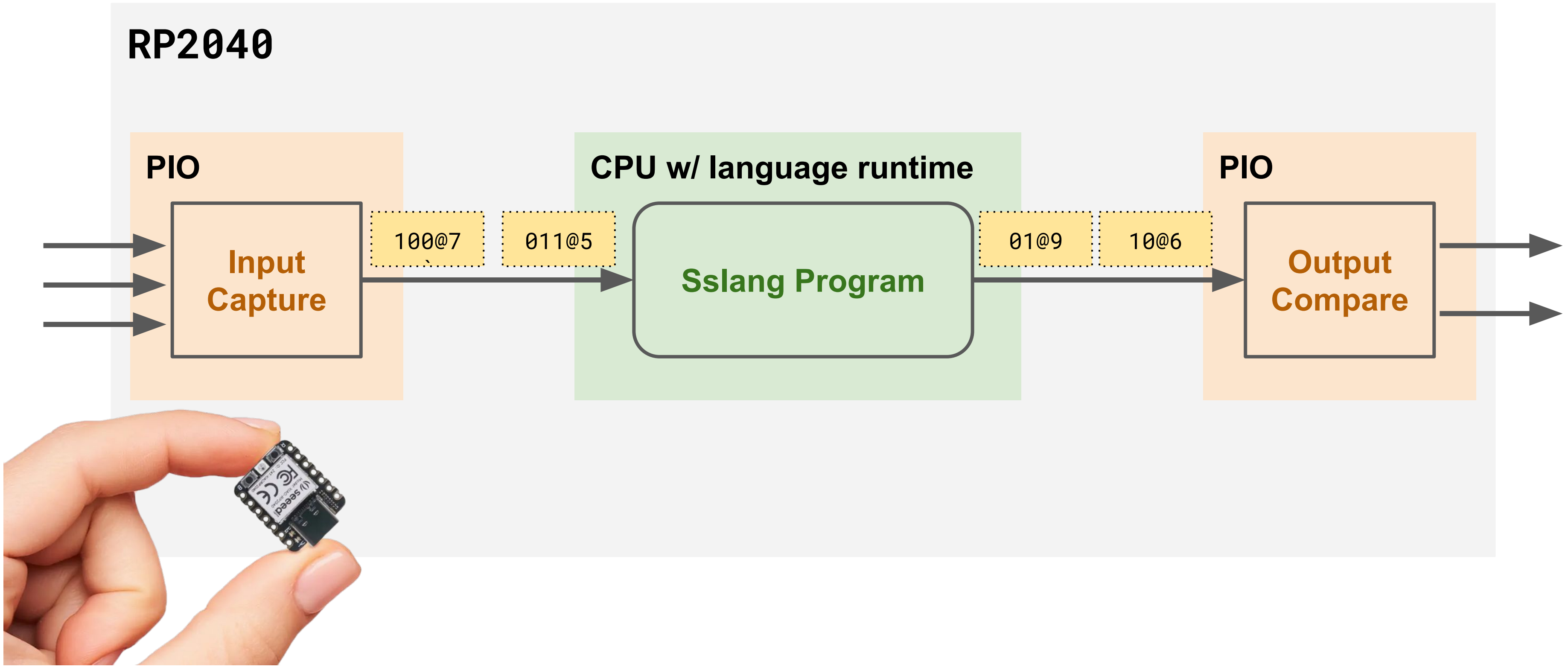
```
blink (press: &()) (led: &Led) =
  loop
    wait press                      // block until button press
    led <- On                       // turn LED on immediately
    after ms 200, led <- Off        // schedule LED off after 200ms
    wait led                        // block until LED turns off
```



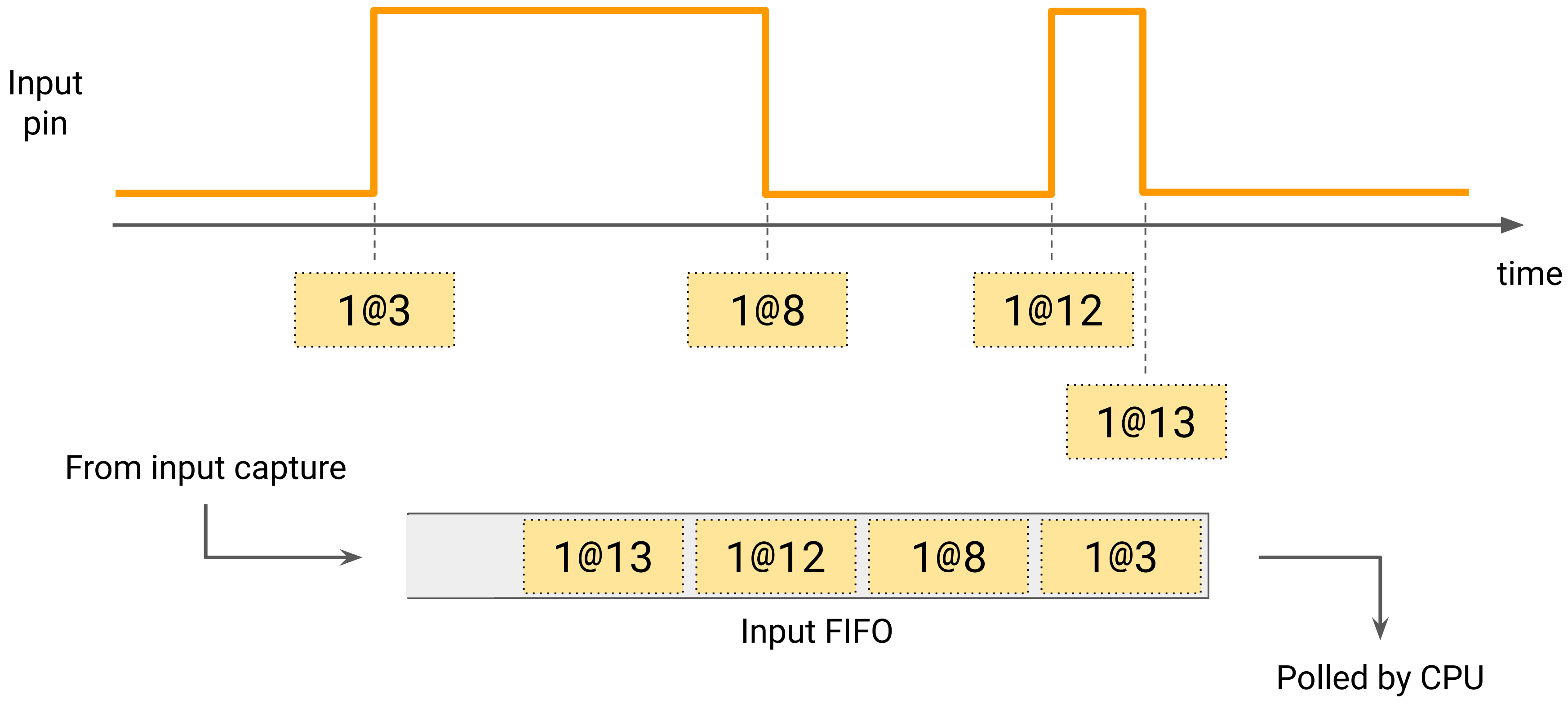
```
debounce (button: &PushButton) (press: &()) =
  loop
    waitfor button Pressed      // active-low button pressed
    press <- ()                 // send "press" event
    sleep (ms 10)               // debounce press
    waitfor button Released     // button released
    sleep (ms 10)               // debounce release
```



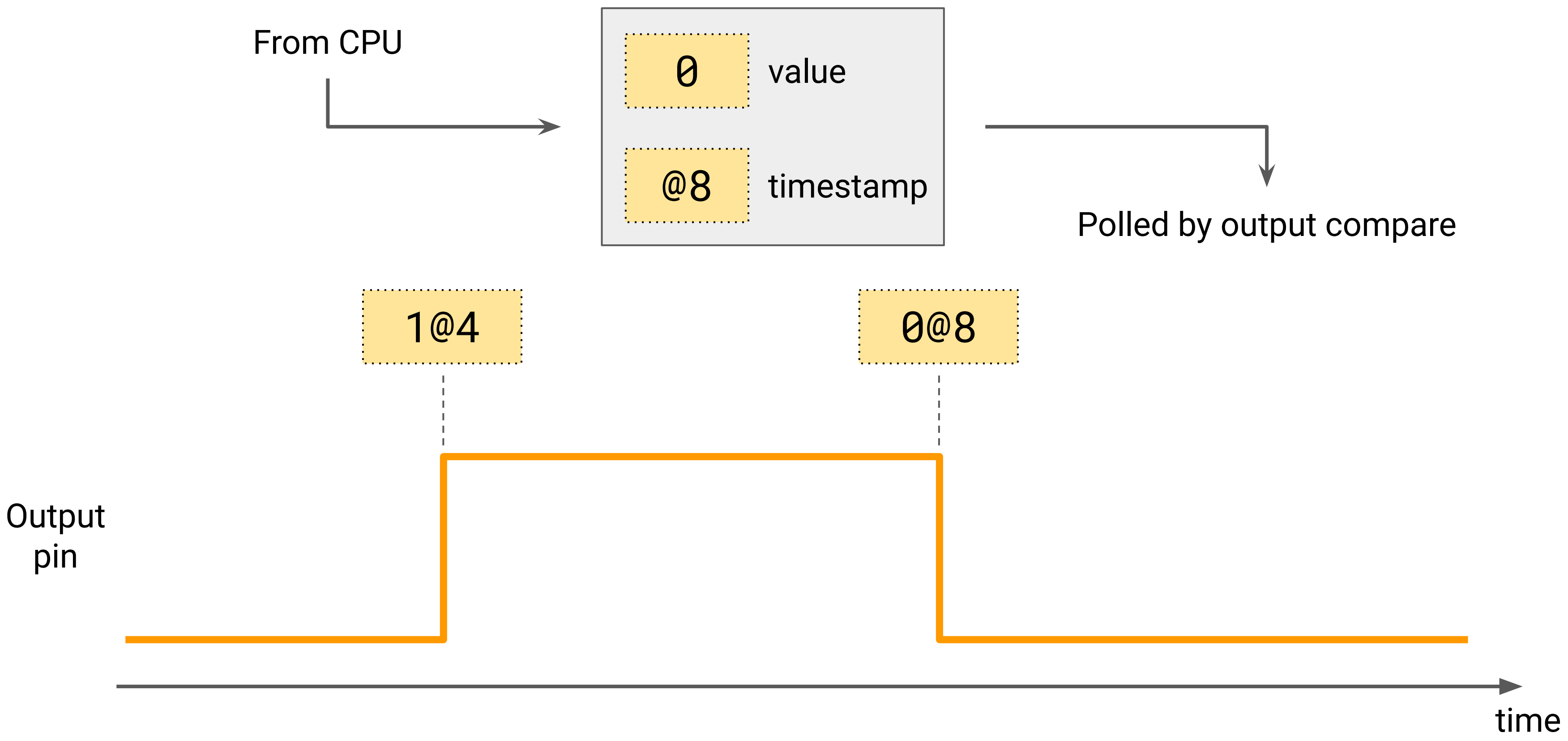
```
blink (press: &()) (led: &Led) =
  loop
    wait press                         // block until button press
    led <- On                          // turn LED on immediately
    after ms 200, led <- Off           // schedule LED off after 200ms
    wait led                           // block until LED turns off

debounce (button: &PushButton) (press: &()) =
  loop
    waitfor button Pressed             // active-low button pressed
    press <- ()                        // send “press” event
    sleep (ms 10)                      // debounce press
    waitfor button Released            // button released
    sleep (ms 10)                      // debounce release

main (button: &PushButton) (led: &Led) =
  let press = new ()
  par debounce button press            // run debounce and blink
      blink    press  led              // in parallel
```


RP2040
PIO
Input Capture
100@7
011@5
CPU w/ language runtime
Sslang Program
01@9
10@6
PIO
Output Compare

Input Capture



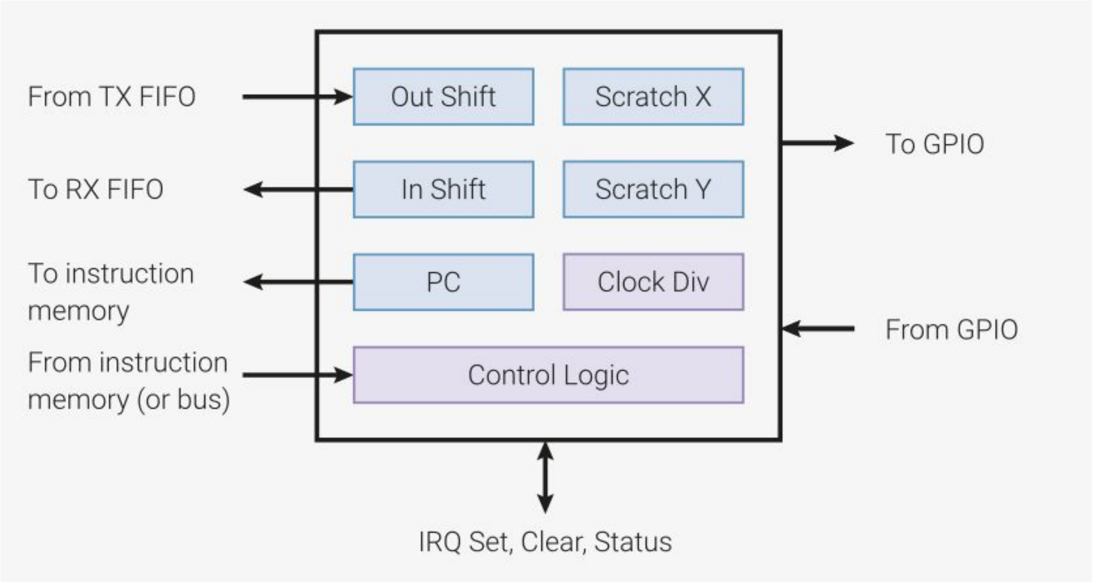
Output Compare



PIO: Programmable I/O

On each PIO device:

- 4 "state machines"
- 32 instruction memory
- 9 op codes
- 4 registers
- Single-cycle execution

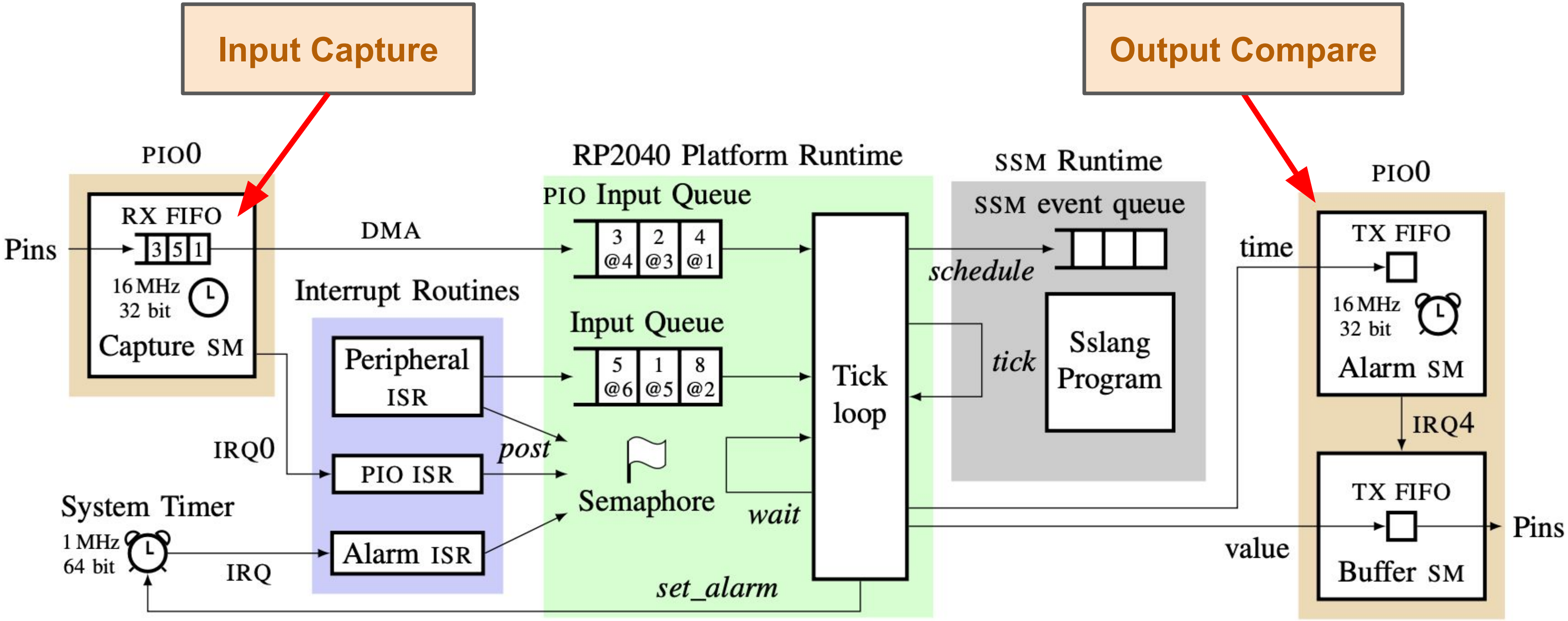


Limited programmability

Clocked using system clock, derived from external crystal oscillator

Timestamp Peripherals

System clock @ 128MHz / 8-cycle counter = **PIO sample rate @ 16MHz**



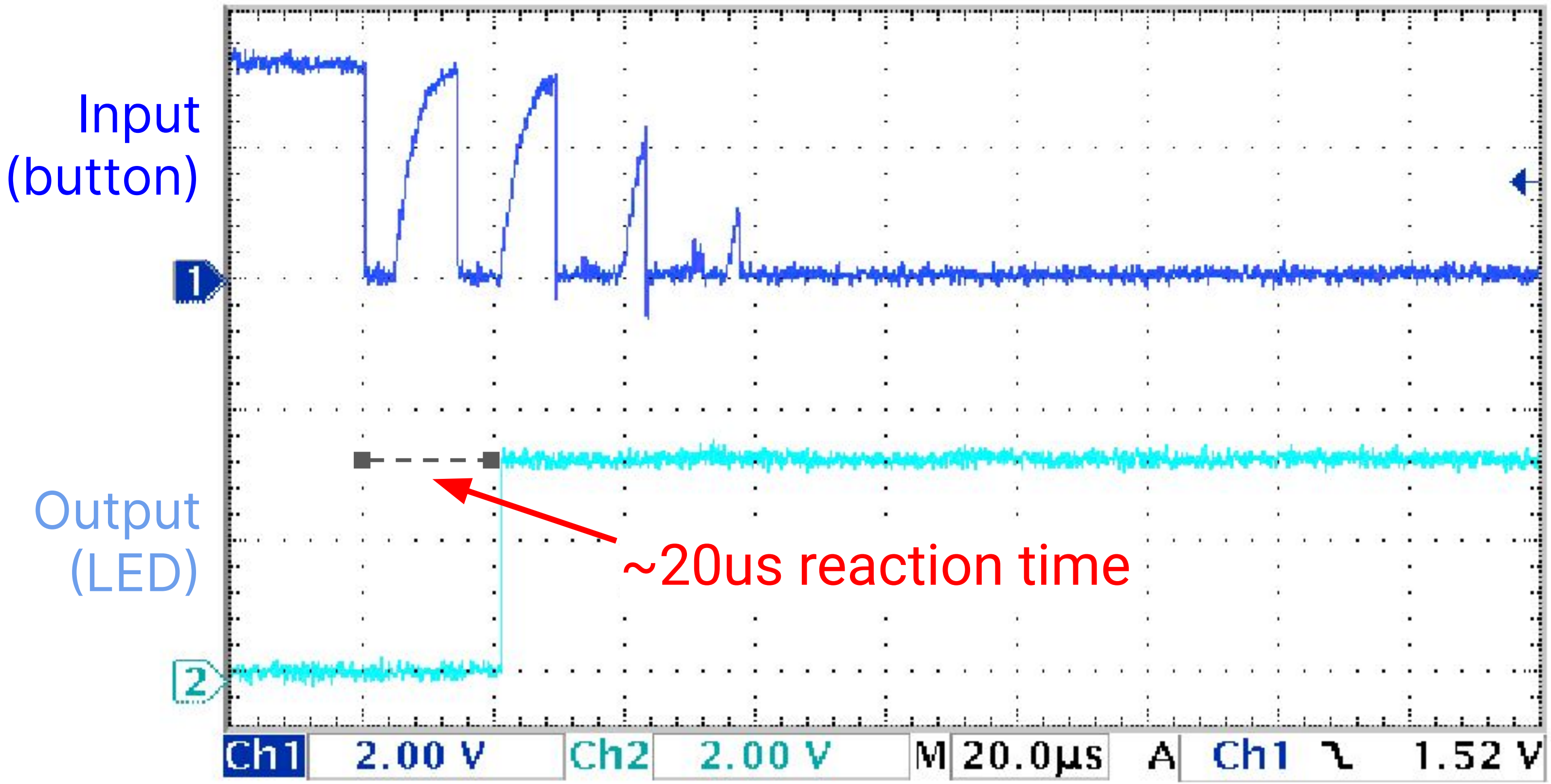
Experimental Goals

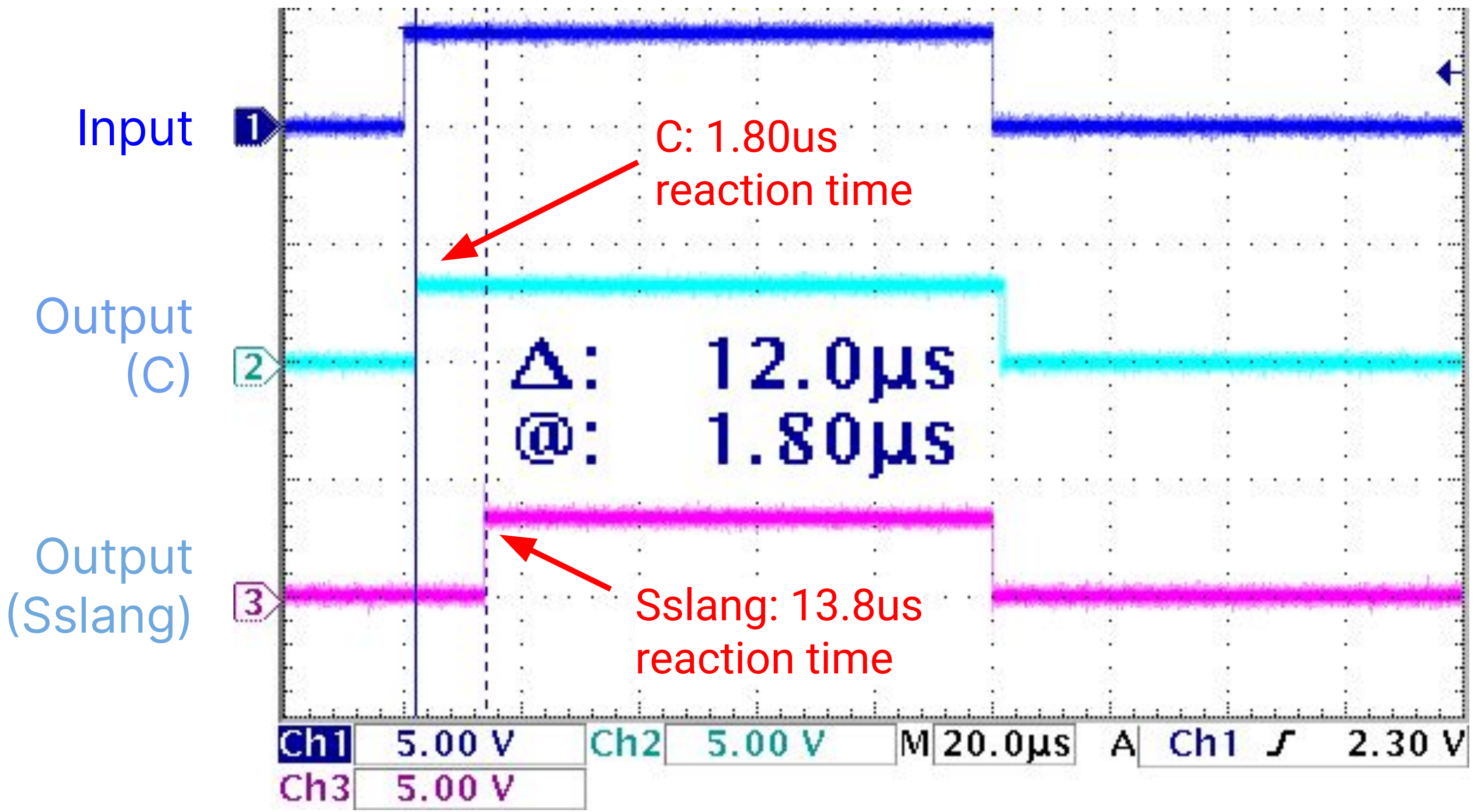
What is the **overhead** of processing events through this system?

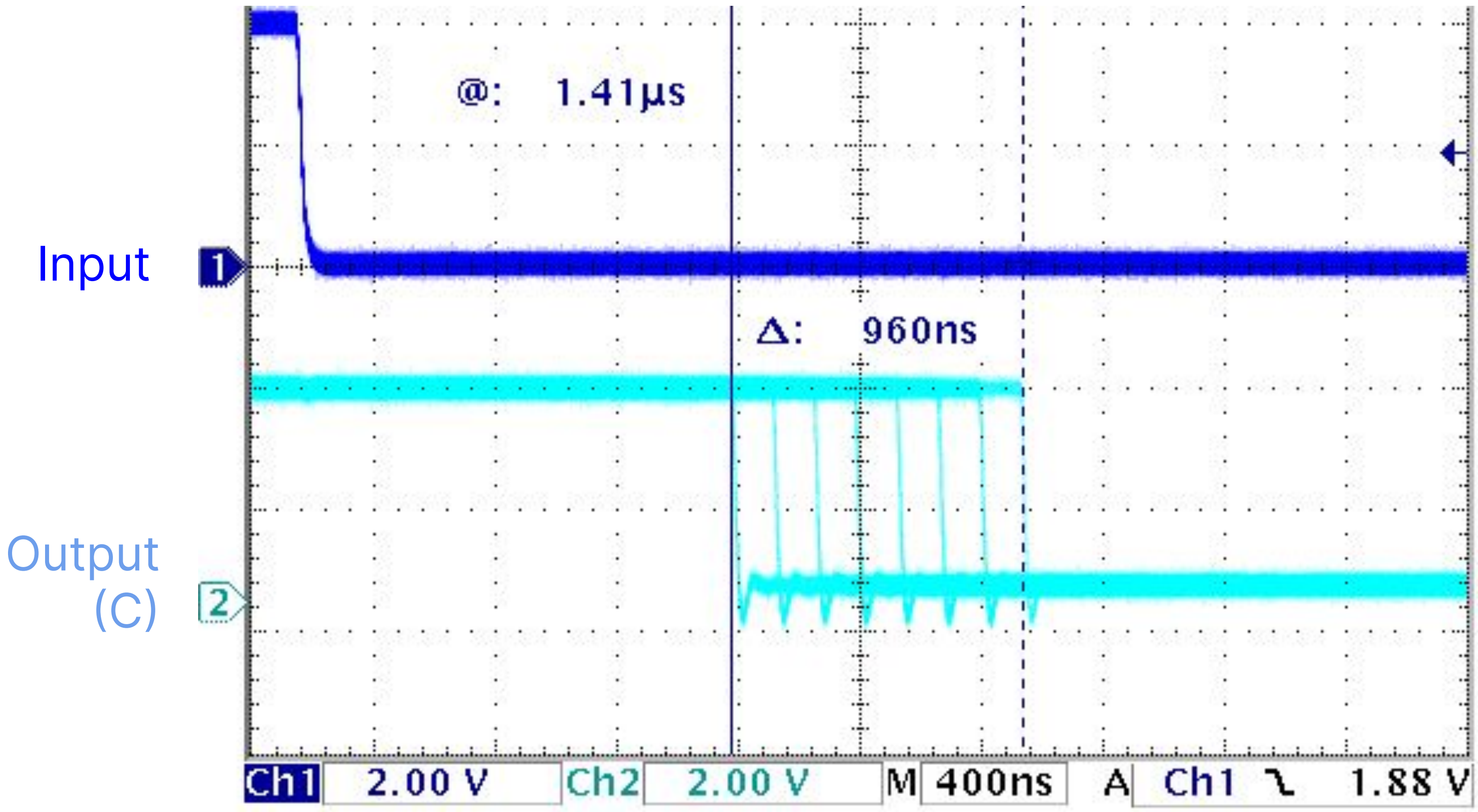
10-20 us

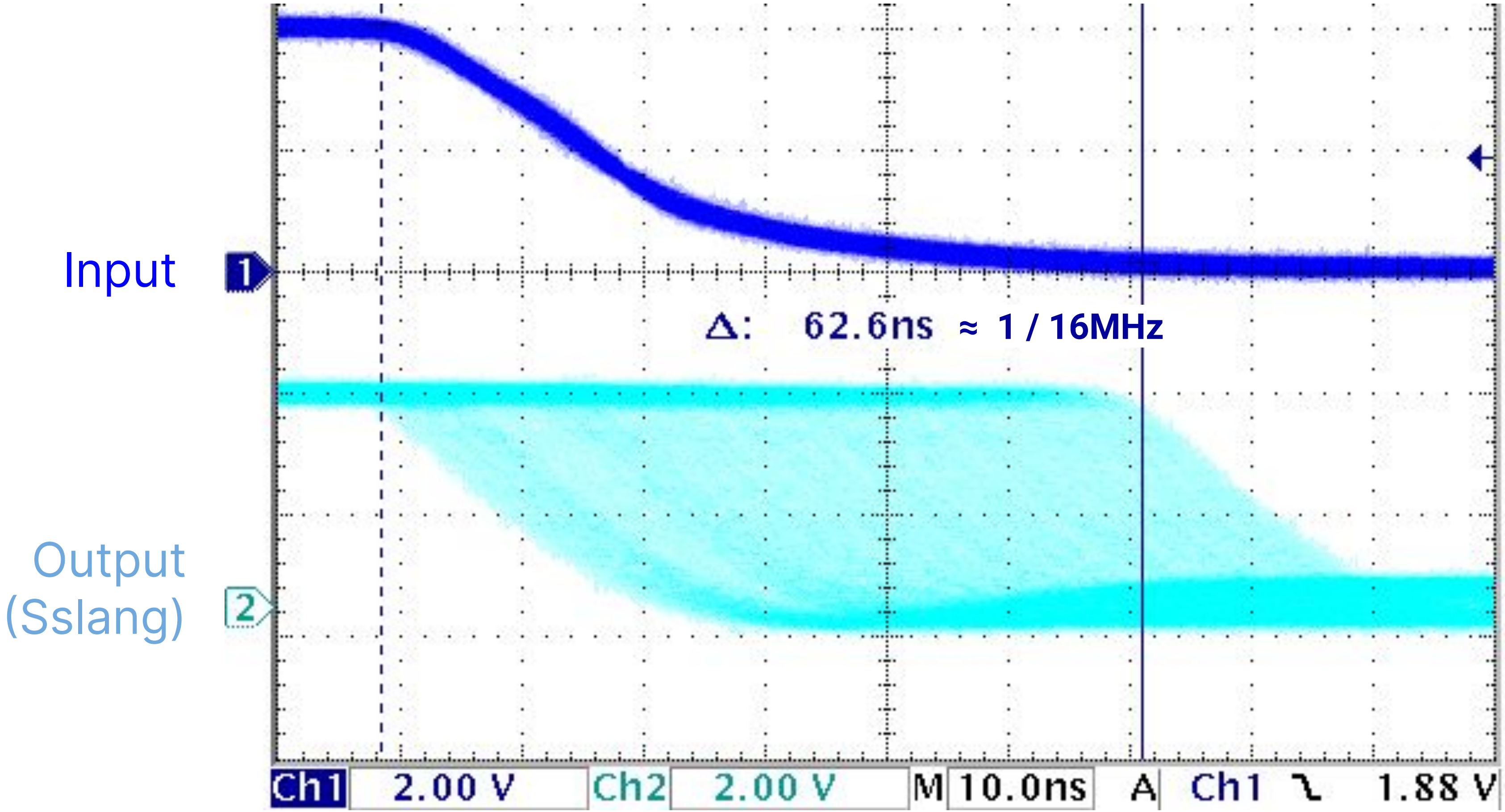
What level of **accuracy** and **precision** can we achieve with timestamp peripherals?

62.5 ns / 16 MHz


Input
(button)
Output
(LED)
~20us reaction time
1
2
Ch1 2.00 V
Ch2 2.00 V
M 20.0µs
A Ch1 1.52 V


Input
Output
(C)
Output
(Sslang)
C: 1.80us
reaction time
Δ: 12.0µs
@: 1.80µs
Sslang: 13.8us
reaction time
Ch1 5.00 V
Ch2 5.00 V
M 20.0µs
A Ch1 2.30 V
Ch3 5.00 V


Input
Output
(C)
1
2
@: 1.41µs
Δ: 960ns
Ch1 2.00 V
Ch2 2.00 V
M 400ns
A Ch1 1.88 V



Input
Output
(Sslang)
Δ: 62.6ns ≈ 1 / 16MHz
Ch1 2.00 V
Ch2 2.00 V
M 10.0ns
A Ch1 1.88 V

Pulse Width Measurement

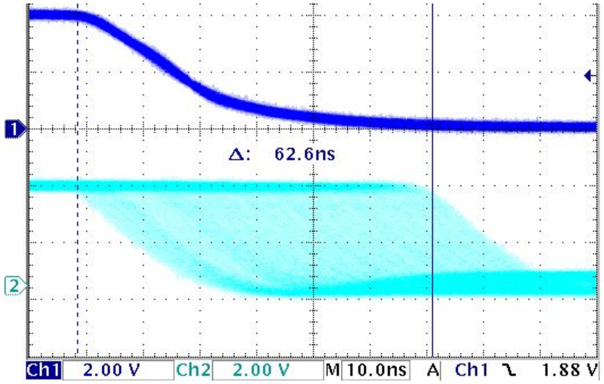
Pulse Input	Expected	Observed	Jitter	Error
80 ms	1 280 000	1 280 021	1	22
8 ms	128 000	128 002	1	3
800 µs	12 800	12 800	1	1
80 µs	1 280	1 280	1	1
8 µs	128	128	1	1
800 ns	12.8	13	1	0.2
80 ns	1.28	2		0.72
40 ns	0.64	2		1.36

Frequency Counter

Frequency	Expected Events	Observed Events
30 kHz	60000	60000
40 kHz	80000	74271
50 kHz	100000	72670
60 kHz	120000	71390
70 kHz	140000	70013
80 kHz	160000	68574
>90 kHz	180000	unstable

```
blink (press: &()) (led: &Led) =
  loop
    wait press
    led <- On
    after ms 200, led <- Off
    wait led
```

SSM + timestamp peripherals



Timestamp peripherals enable **precise timing** behavior from **expressive** synchronous languages

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

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