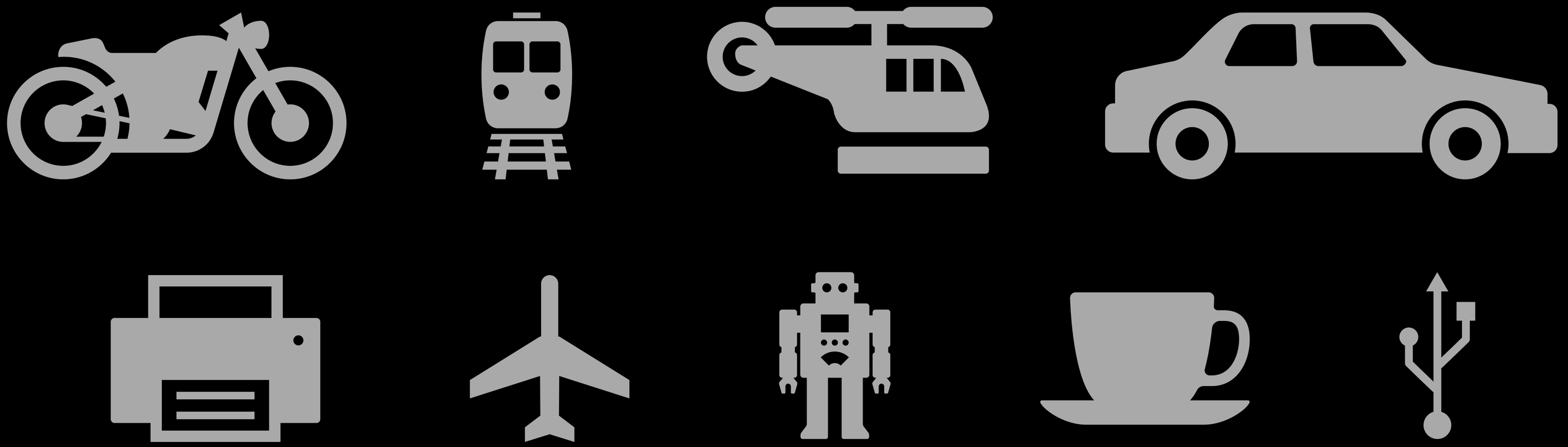
Pipit: reactive systems in F*

Amos Robinson & Alex Potanin
Australian National University (ANU)

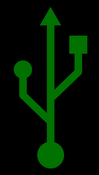
TyDe 2023

Reactive systems



Sensor driver

display

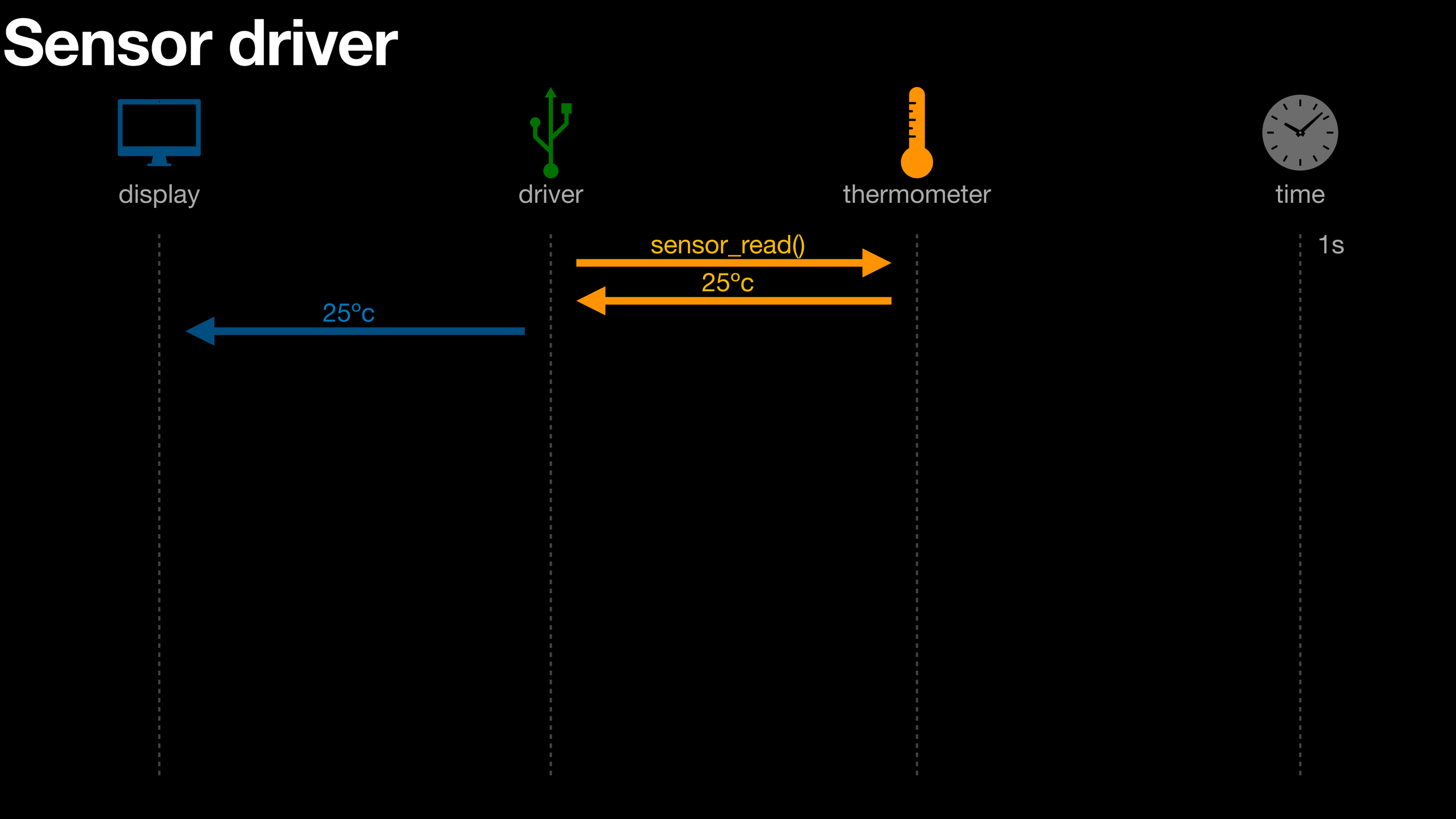


driver

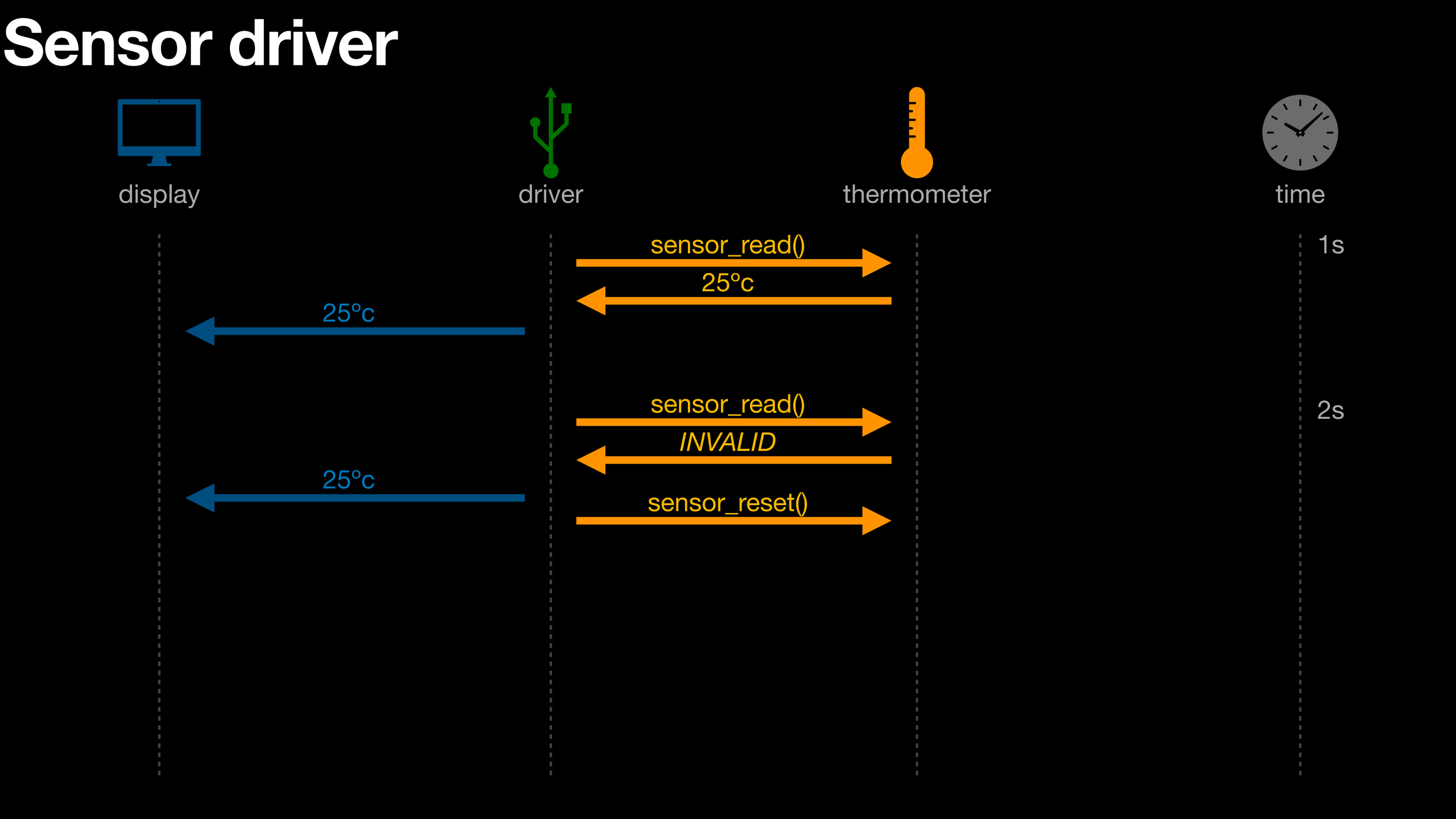
thermometer

time

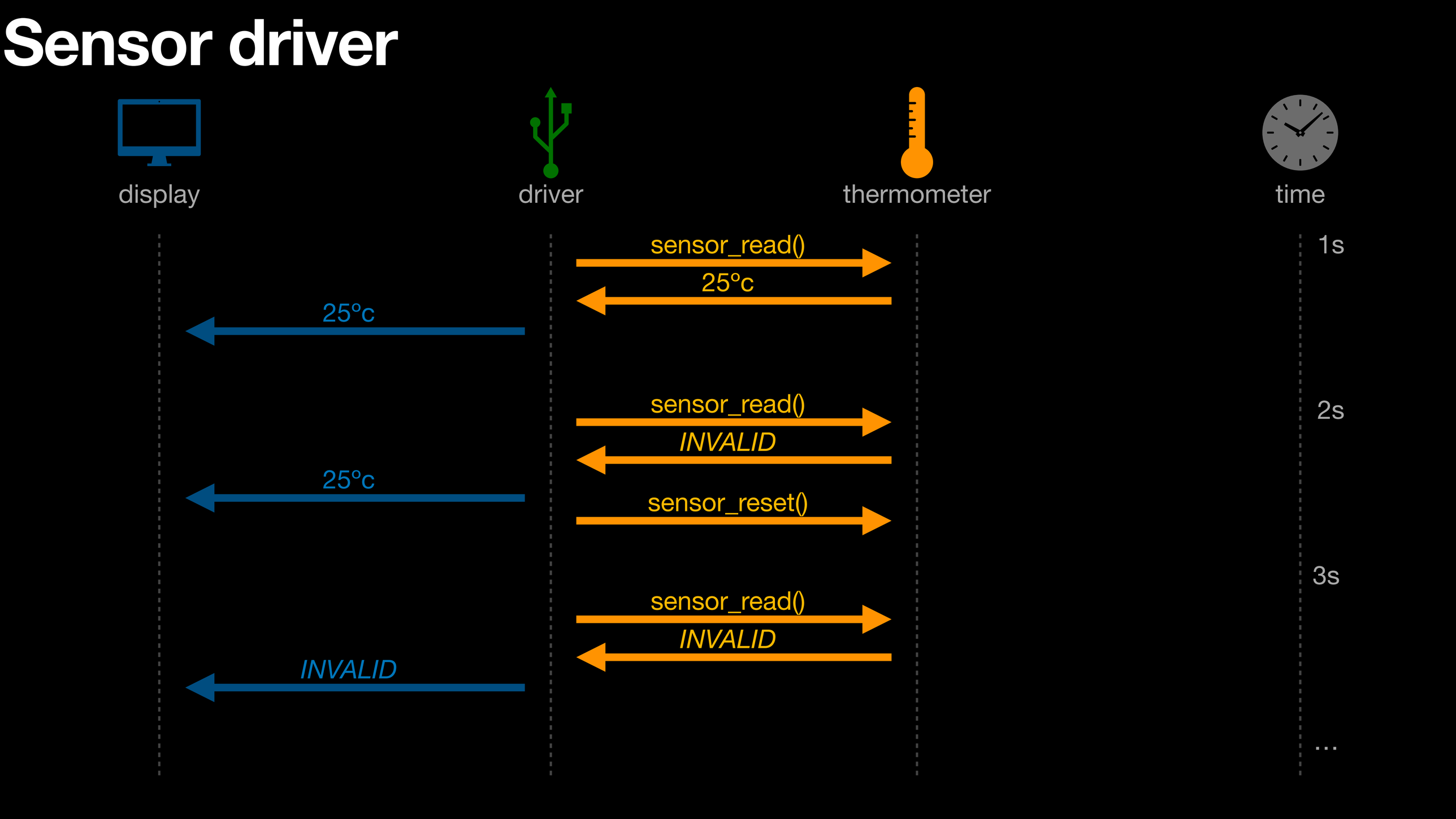
Sensor driver



Sensor driver



Sensor driver



Sensor driver: implementation

```
sensor_driver() {
    int    age    = VALID_TIMEOUT;
    sample status = SAMPLE_INVALID;

    while (true) {
        sample s = sensor_read();

        if (SAMPLE_VALID(s)) {
            status = s;
            age    = 0;
        } else {
            age  = age + 1;
        }

        if (age >= VALID_TIMEOUT) {
            status = SAMPLE_INVALID;
        }
        update_status(status);

        if (age == RESET_TIMEOUT) {
            sensor_reset();
        }
        sleep(1000ms);
    }
}
```

Sensor driver: synchronous split

```
sensor_driver() {
    sensor_logic_state state;
    sensor_logic_init(&state);

    while (true) {
        sample s = sensor_read();

        sensor_logic_result result;
        result   = sensor_logic_step(s, &state);

        update_status(result.status);

        if (result.reset) {
            sensor_reset();
        }

        sleep(1000ms);
    }
}
```

```
sensor_logic (sensor_read: stream sample):
    stream { status: sample
           ; reset:  bool }
```

Sensor driver: synchronous split

```
sensor_driver() {
    sensor_logic_state state;
    sensor_logic_init(&state);

    while (true) {
        sample s = sensor_read();

        sensor_logic_result result;
        result   = sensor_logic_step(s, &state);

        update_status(result.status);

        if (result.reset) {
            sensor_reset();
        }

        sleep(1000ms);
    }
}
```

```
sensor_logic (sensor_read: stream sample):
    stream { status: sample
           ; reset:  bool } =

  let rec age =
    if SAMPLE_VALID sensor_read then
      0
    else
      (VALID_TIMEOUT `fby` age + 1)
  in
  let rec status =
    if SAMPLE_VALID sensor_read then
      sensor_read
    else if age < VALID_TIMEOUT then
      (SAMPLE_INVALID `fby` status)
    else
      SAMPLE_INVALID
  in
  let reset =
    (age == RESET_TIMEOUT)
  in
    { status = status; reset = reset }
```

Catching bugs

Sensor driver: partial specification

```
sensor_logic (sensor_read: stream sample):
    stream { status: sample
           ; reset:  bool }

spec (sensor_read: stream sample): prop =
  let { status, reset } = sensor_logic sensor_read
  in
    forall t. SAMPLE_VALID (status @ t) ==>
    exists t' <= t.
      (status @ t = sensor_read @ t')
```

Sensor driver: partial specification

```
sensor_logic (sensor_read: stream sample):
    stream { status: sample
           ; reset:  bool }

spec (sensor_read: stream sample): prop =
  let { status, reset } = sensor_logic sensor_read
  in
    forall t. SAMPLE_VALID (status @ t) ==>
    exists t' <= t.
      (status @ t = sensor_read @ t' /\
        forall t''. t' < t'' < t ==> not (SAMPLE_VALID (sensor_read @ t'')))
```

Sensor driver: synchronous specification

```
sensor_logic (sensor_read: stream sample):
    stream { status: sample
           ; reset:  bool }

spec (sensor_read: stream sample): stream bool =
  let { status, reset } = sensor_logic sensor_read
  let rec latest =
    if SAMPLE_VALID sensor_read then
      sensor_read
    else
      (SAMPLE_INVALID `fby` latest)
  in
    SAMPLE_VALID status ==> status = latest
```

Sensor driver: synchronous specification

```
spec (sensor_read: stream sample): stream bool =
  let { status, reset } = sensor_logic sensor_read
  let rec latest =
    if SAMPLE_VALID sensor_read then
      sensor_read
    else
      (SAMPLE_INVALID `fby` latest)
  in
    SAMPLE_VALID status ==> status = latest

sys: system sample bool =
  system_of_exp (run1 spec))

valid: system_valid sys =
  assert (system_valid_induct_1_base sys)
    by normalize_system;
  assert (system_valid_induct_1_step sys)
    by normalize_system;

  assert (system_valid_induct_1 sys)
```

Sensor driver: synchronous specification

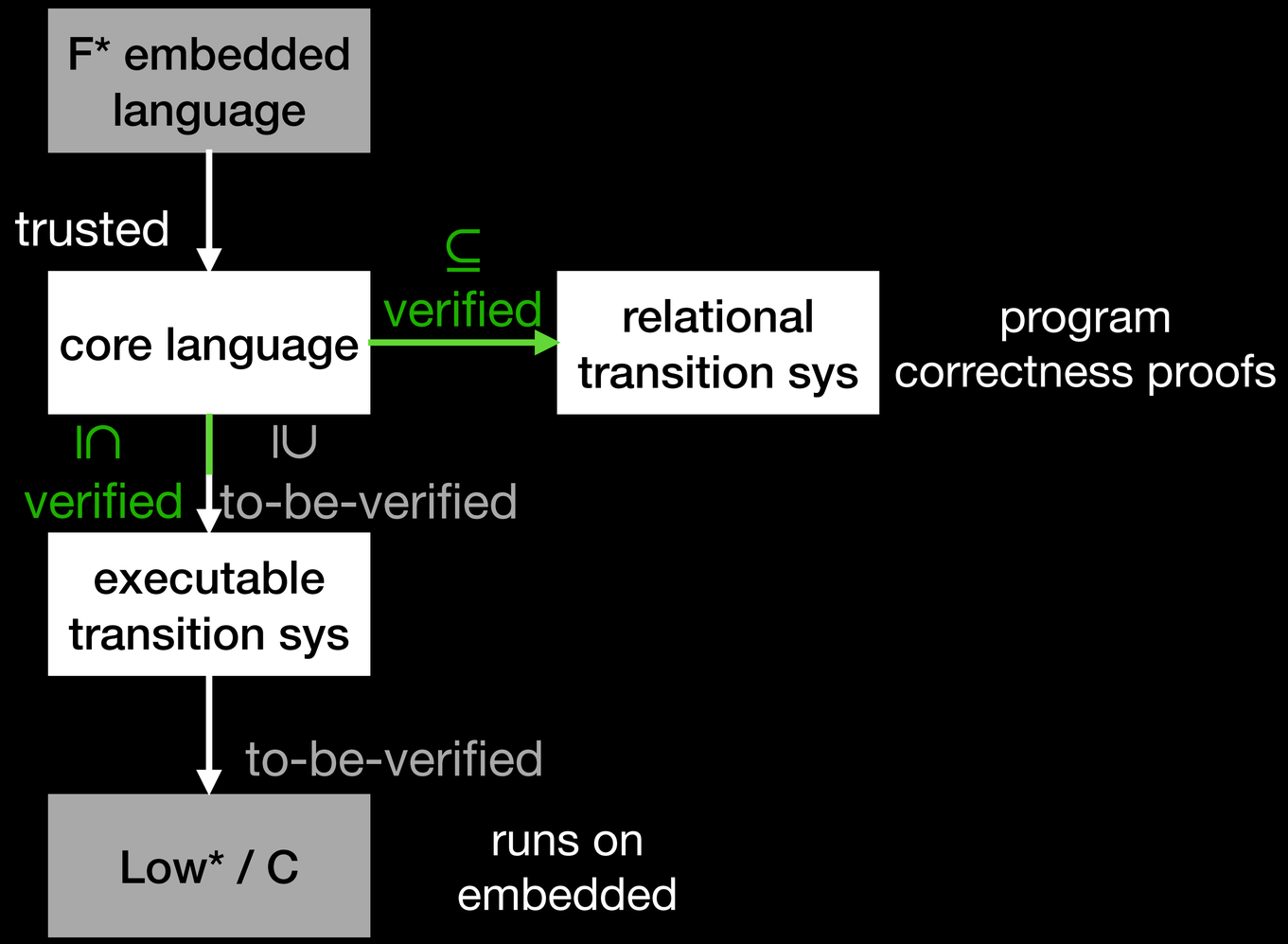
```
spec (sensor_read: stream sample): stream bool =
  let { status, reset } = sensor_logic sensor_read
  let rec latest =
    if SAMPLE_VALID sensor_read then
      sensor_read
    else
      (SAMPLE_INVALID `fby` latest)
  in
    SAMPLE_VALID status ==> status = latest

sys: system sample bool =
  system_of_exp (run1 spec))

==normalize_system==>

  system {
    init = (max_age, SAMPLE_INVALID, SAMPLE_INVALID);
    step = fun read (age, status, latest) (age', status', latest') ok.
       age'    = (if SAMPLE_VALID read then 0 else age + 1)
    /\ status' = (if SAMPLE_VALID read then read else if age' < MAX_AGE then status else SAMPLE_INVALID)
    /\ latest' = (if SAMPLE_VALID read then read else latest)
    /\ ok      = (SAMPLE_VALID status' ==> status' = latest');
  }
```

Pipit

Core language

$$
\begin{aligned}
e :=& \\
& \mid v \mid x \mid p(e\ldots) \\
& \mid e \text{ fby } e' \\
& \mid \text{rec } x.\ e[x] \\
& \mid \text{let } x = e \text{ in } e'[x] \\
& \mid \text{check } e
\end{aligned}
$$

Pipit structure



Conclusion and future work

Now:

- Partially-verified programs on real µcs

Next:

- Rely-guarantee contracts
- Common subexpression elimination
- Evaluation:
  - More drivers, "higher-order" drivers

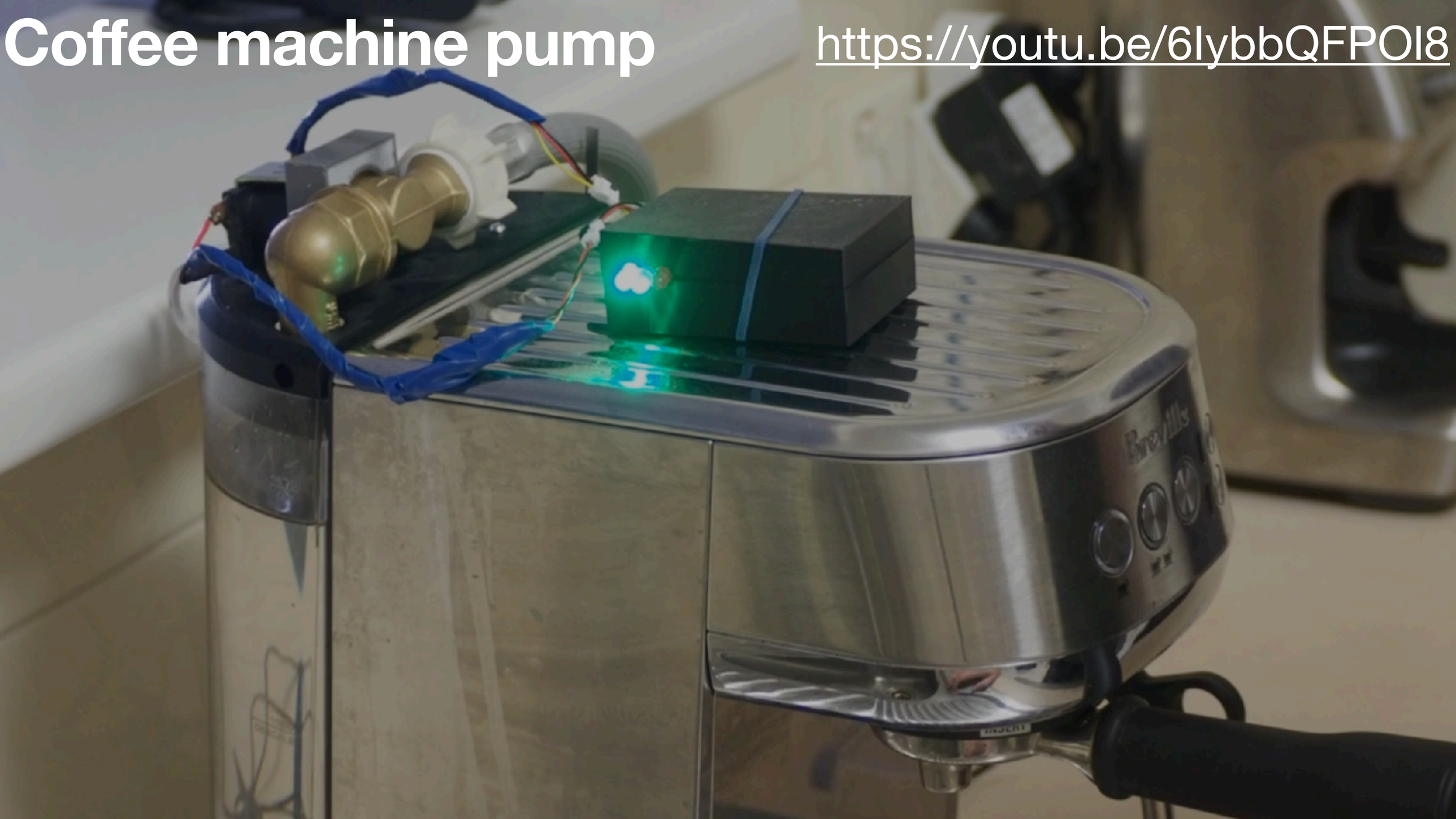
Later:

- Clocks for more expressive streaming

Appendices

Coffee machine pump

https://youtu.be/6IybbQFPOI8



Transition systems

```
type system_rel i o = {
  state: Type;
  init:  state → 𝔹;
  step:  i → state → state → o → 𝔹;
}

type system_exec i o = {
  state: Type;
  init:  state;
  step:  i → state → (state × o);
}
```

Trusted computing base

Pipit (compile+check)

- OCaml
- F*
- Z3
- Karamel (F*->C)

- + remaining proofs…
- + C compiler

Vélus (compile) + Kind2 (check)

- OCaml
- Coq

- +CompCert TCB

- OCaml
- Z3
- Lustre -> SMT

- *certificates