Lustre (or other synchronous languages) for Arduino

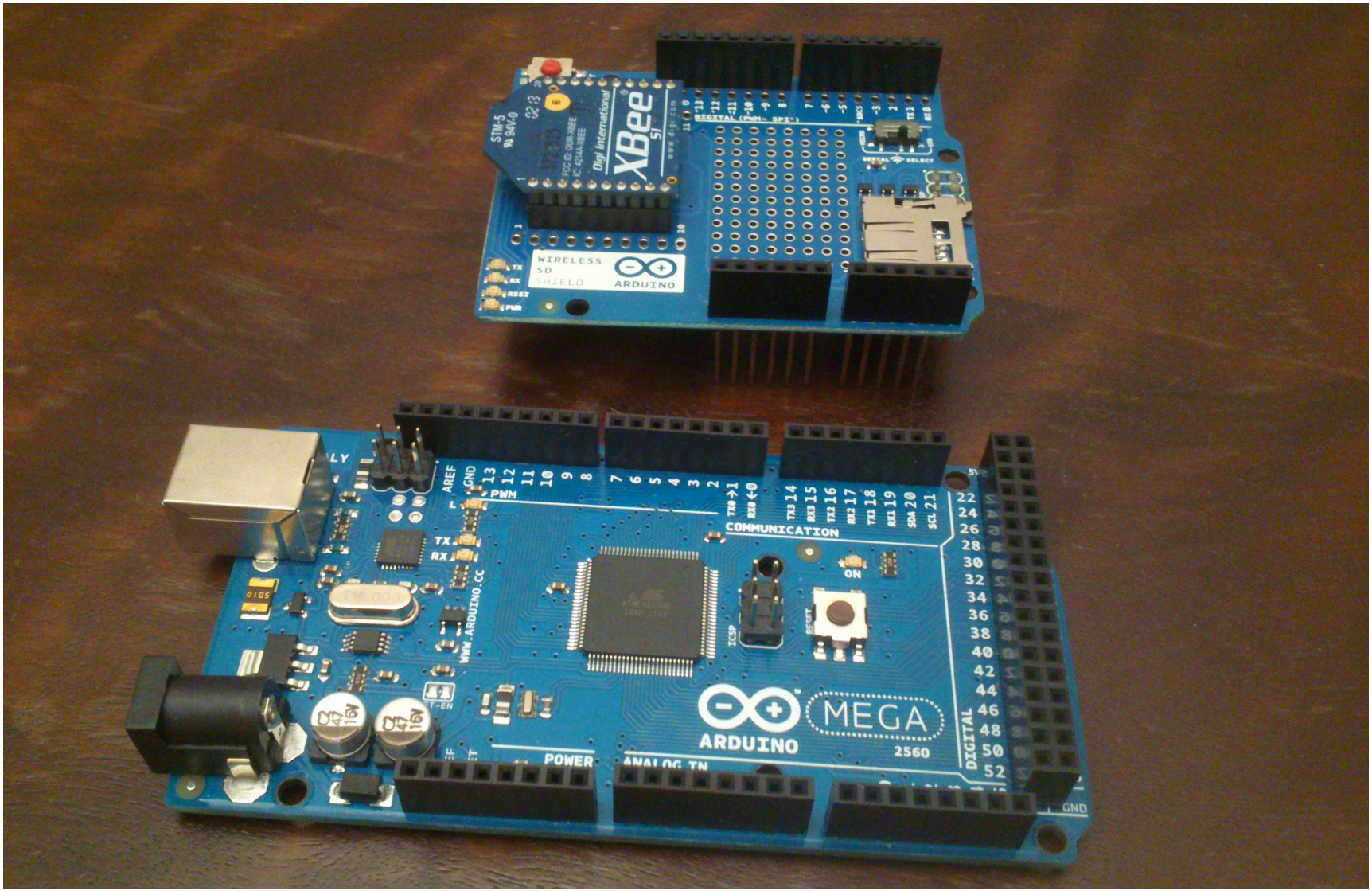
H.-J. Audéoud, F. Maraninchi[1],
Grenoble INP/Ensimag and VERIMAG

SYNCHRON'14, Aussois

December 4, 2014

[1] http://orcid.org/0000-0003-0783-9178

Arduino (1)


XBee
WIRELESS
SD
SHIELD
ARDUINO
COMMUNICATION
POWER
ANALOG IN
DIGITAL
ARDUINO
MEGA
2560

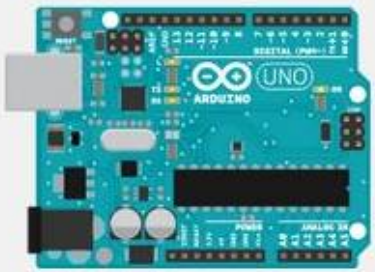
Arduino (2) - See www.arduino.cc

ARDUINO

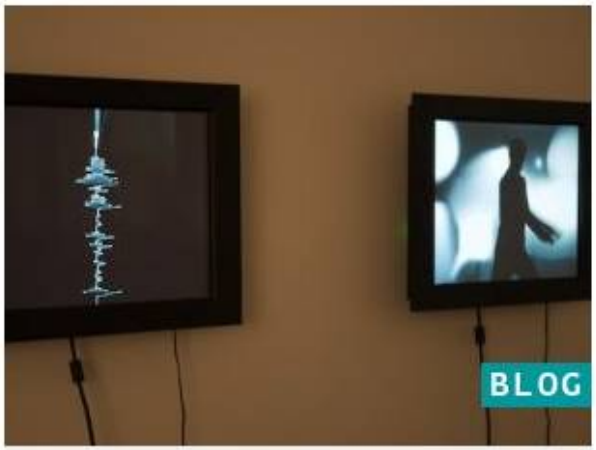
Search the Arduino Website

Home Buy Download Products Learning Forum Support Blog LOG IN SIGN UP

WHAT IS ARDUINO?



BUY AN ARDUINO



EARTHQUAKES

REINTERPRETED BY THE

ARDUINO MATERIA 101,

SIMPLIFYING 3D

PRINTING.

PRE-ORDER NOW

Arduino (3) - video

A safety-critical example, by Henri-Joseph.

Example Program - arduino.cc/en/Tutorial/Button

```
const int buttonPin = 2;
  // the number of the pushbutton pin
const int ledPin =  13;
  // the number of the LED pin

int buttonState = 0;
  // variable for reading the pushbutton status

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}
```

Example Program - arduino.cc/en/Tutorial/Button

```
void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);
  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

The same in Lustre (no memory needed)

```
node Button (button: bool) returns (ledcmd: bool) ;
let
   ledcmd = button ;
tel.
```

+ compiler into C
+ usual bla bla for the interfacing of the sensors/actuators

The C code produced by the excellent Raymond's compiler

```
#include "Button.h"
typedef struct  {
   //INPUTS
   _boolean _button;
   //OUTPUTS
   _boolean _ledcmd;
   //REGISTERS
} Button_ctx;
static Button_ctx ctx;
```

The C code produced by the excellent Raymond's compiler

```
// input function
void Button_I_button(_boolean V){
   ctx._button = V;
}
// Output function
extern void Button_O_ledcmd(_boolean);
// Reset procedure
void Button_reset(){...}
// Step procedure
void Button_step(){ Button_O_ledcmd(ctx._button);}
```

The Main Loop

```
... buttonState ;
const int buttonPin = 2; const int ledPin = 13;
setup()
Button_reset () ;
while (1) {
  buttonState = digitalRead(buttonPin); // Arduino style
  Button_I_button(buttonState==HIGH); // Lustre
  Button_step ()
   // in which the output procedures are called
   // e.g., Button_O_ledcmd(_boolean);
```

First Limitation of the Arduino Programming Model... and solution

If there are a lot of inputs and outputs, or if the behavior you need has memory, the body of the loop may become complex, and it's error-prone to write it by hand.

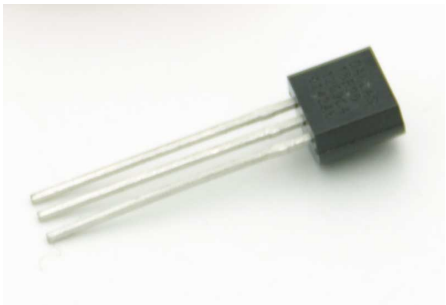
Lustre is the answer: think parallel, get the loop code for free

Example with Memory

```
node Button (button: bool) returns (ledcmd: bool) ;
var number_pressed : int ;
let
   number_pressed =
         0 -> (pre number_pressed +
               (if button then 1 else 0))
              mod 42 ;
   ledcmd = number_pressed < 12 ;
tel.
```

The DS18B20 Temperature Sensor and the OneWire Protocol

http://playground.arduino.cc/Learning/OneWire



On a 1-Wire network, a single "master" device communicates with one or more 1-Wire "slave" devices over a single data line, which can also be used to provide power to the slave devices.

Example: Typical Use of the DS18B20 Temperature Sensor

```
#include <OneWire.h> // One Wire protocol
OneWire ds(DS18B20_pin);

// Start the temperature measurement
ds.reset();ds.skip();ds.write(0x44, 1);

// Wait until the measure is available
delay(750);
  // in milliseconds, not kgs or truckloads of bananas

// Retrieve the value
ds.reset();ds.skip();ds.write(0xBE);
temperature = (ds.read() + (ds.read() << 8)) * 0.0625;
```

Known Problem: the Delay

http://playground.arduino.cc/Learning/OneWire

The majority of existing code for 1Wire devices, particularly that written for Arduino, uses a very basic "Convert, Wait, Read" algorithm, even for multiple devices.

Program timing for other functions:
Arguably the biggest problem with using the above methodology is that unless threading measures are undertaken, the device must sit (hang) and wait for the conversion to take place if a hardcoded wait time is included. ... a 12-bit conversion process for a DS18B20 can take as long as 750ms.

Where to Put this Code in the Lustre Version?

- In the `input` function for the temperature sensor, simple, but it means the input operation hides a 750 ms delay
- Partly in the `input` function, and partly in an `output` function

General Solutions for Sensors like that

- Part of the Lustre program is in charge of providing the temperature for the other parts; it produces measuring orders periodically, and reads the sensor periodically too, ensuring the 750 ms delay;

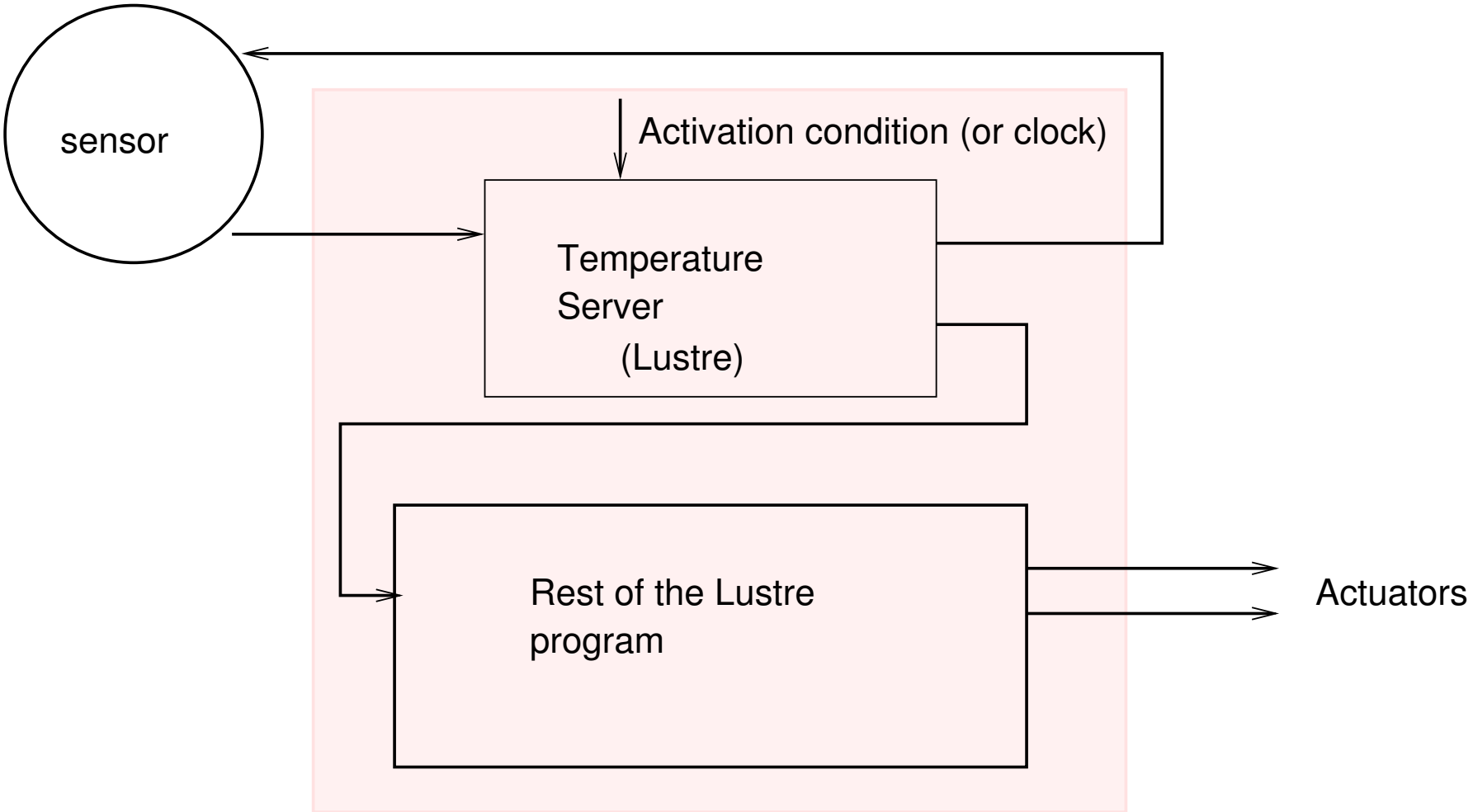
General Solutions for Sensors like that

- Part of the Lustre program is in charge of providing the temperature for the other parts; it produces measuring orders periodically, and reads the sensor periodically too, ensuring the 750 ms delay;
- In order to reduce energy consumption, the temperature could be updated on a clock of, say, 3s only. ...

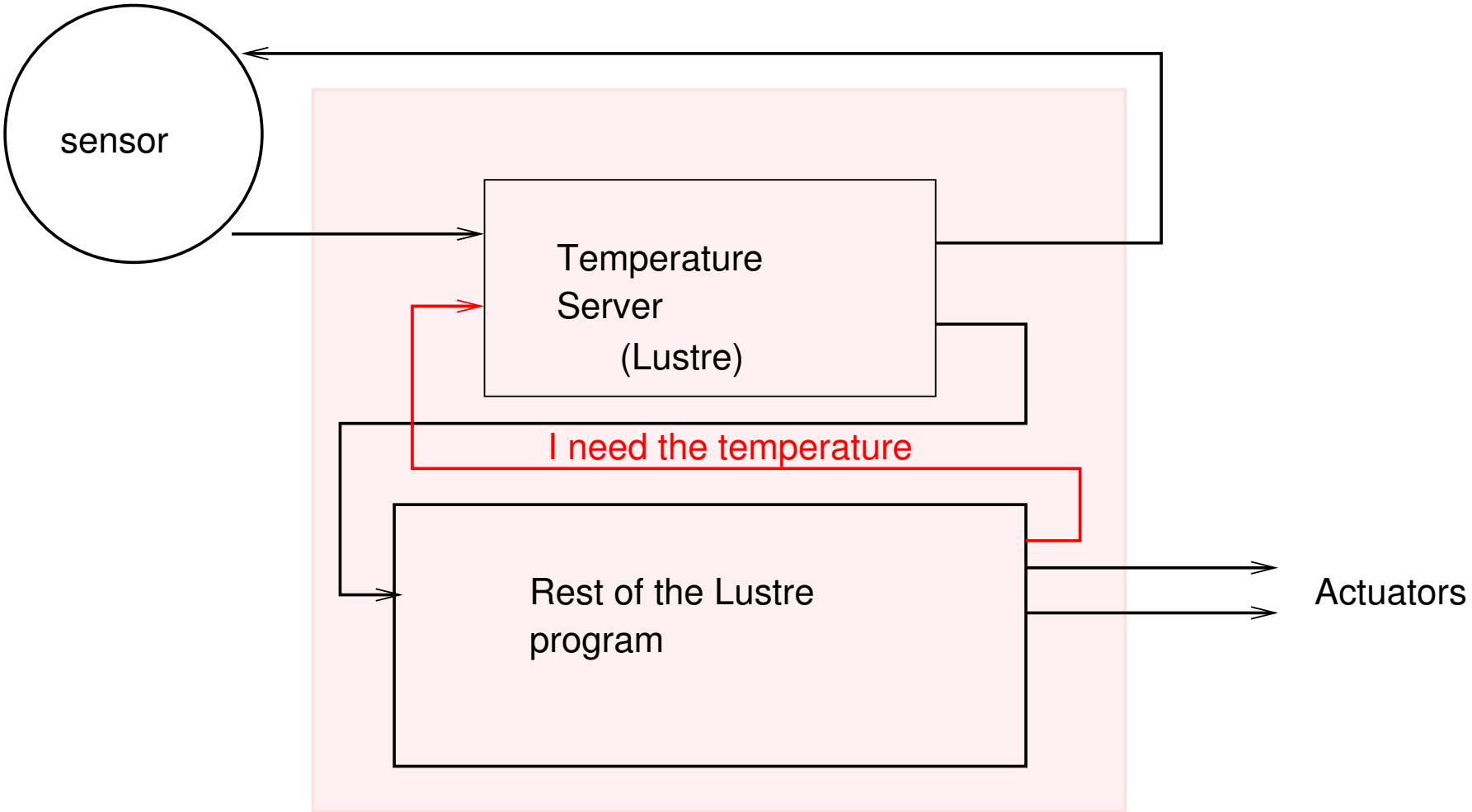
General Solutions for Sensors like that

- Part of the Lustre program is in charge of providing the temperature for the other parts; it produces measuring orders periodically, and reads the sensor periodically too, ensuring the 750 ms delay;
- In order to reduce energy consumption, the temperature could be updated on a clock of, say, 3s only. ...
- The temperature could be updated on demand only. The Lustre program produces measuring orders, but not periodically.

General Solutions for "delay" Sensors (1)



General Solutions for "delay" Sensors (2)



Slight Adaptation of the Main Loop (Clocks on Inputs)

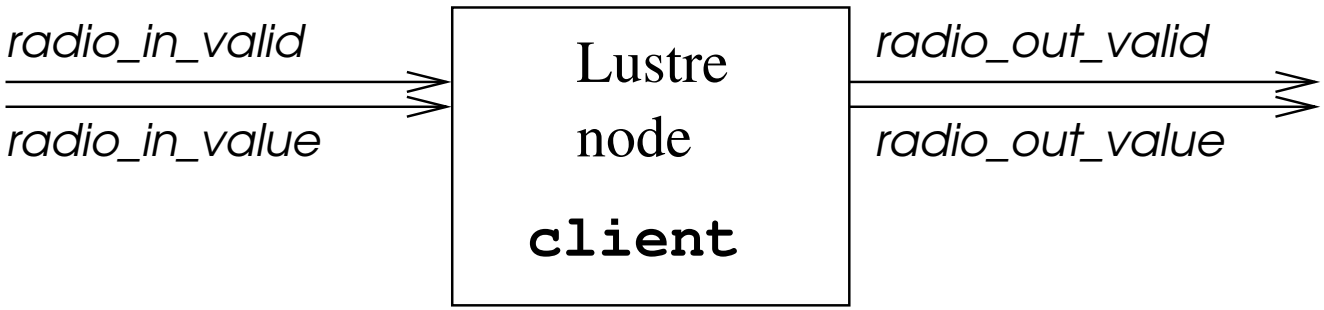
```
bool must_read_input = 0;
reset();
while (1) {
  // use the input procedures for sensors without delay
  if (must_read_input) {
      use the input procedure for temp.
      must_read_input = 0;
  }
  step (...)
   // in which the output procedures are called
   // one of them sets must_read_input
}
```

Radio interface

- An input buffer, storing input data when they are decoded by the radio.
- `Serial.available` returns the number of bytes available in this buffer
- `Serial.read` takes the first byte of it.

- `Serial.write` sends values given as parameter on the serial interface.

In Lustre

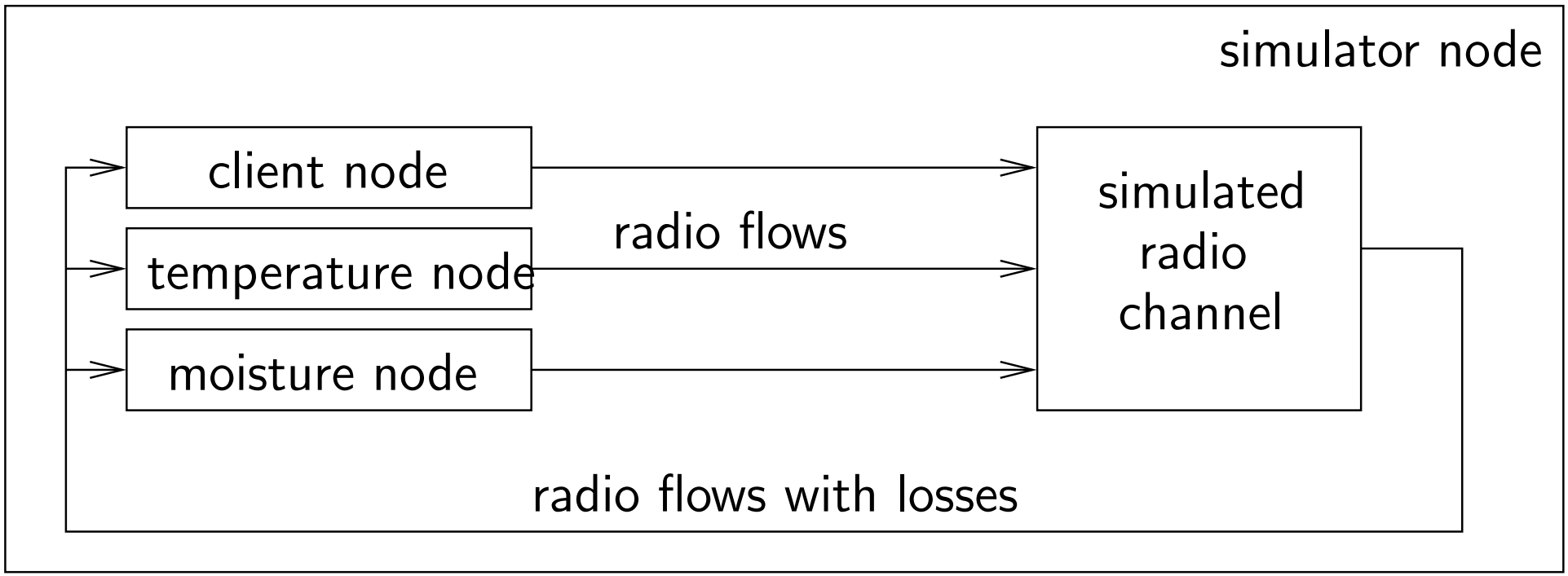

radio_in_valid
radio_in_value
Lustre
node
client
radio_out_valid
radio_out_value

In Lustre, with the main loop

```
void loop() {
  // This fits perfectly in an input procedure,
  // but for a tuple of inputs.
  if (Serial.available > 0) {
    client_I_radio_in_valid(true);
    client_I_radio_in_value(Serial.read());
  } else { // If the buffer is empty
    client_I_radio_in_valid(false);
  }

  client_I_step();
}
```

Complete Simulations, with models of the radio channel



Conclusion

- The Raymond's compiler produces C code that can be used directly for the required `setup` and `loop` functions of the Arduino programming style (with the option `-ctx-static`)
- Some work has to be done on the interfaces for "delay" sensors
- This implies a programming style in Lustre where you set an output when you need a new sample of the input
- The serial interface of the radio is easy to use
- Lurette/Lutin... can be used to simulate a distributed algorithm with the real code of each node

Questions?