FDL '20, September 15 2020, Kiel, Germany

A Language for Deterministic Coordination Across Multiple Timelines

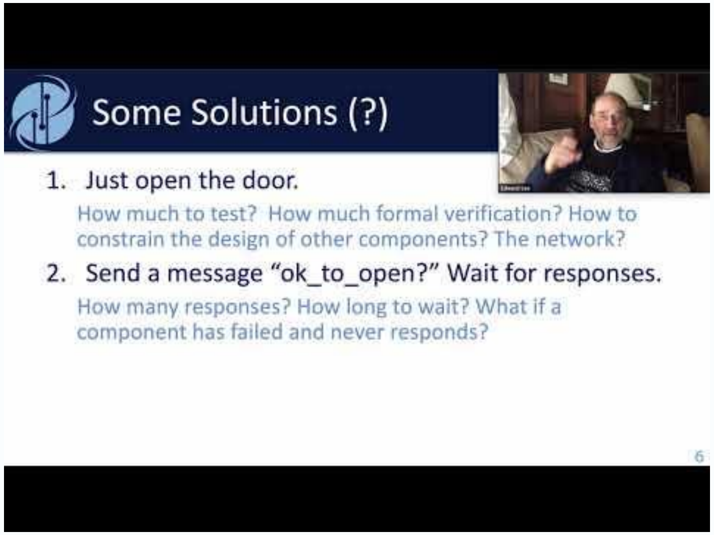
Marten Lohstroh[★], Christian Menard[✪], Alexander Schulz-Rosengarten[★],
Matthew Weber, Jeronimo Castrillon[✪], and Edward A. Lee[★]

[★] Attending remotely

[✪] Attending in person

FDL '20 Keynote on Lingua Franca

For more background, also see Prof. Lee's Keynote Talk on Lingua Franca: https://www.youtube.com/watch?v=_jbdWky4Iys



Some Solutions (?)

1. Just open the door.

   How much to test? How much formal verification? How to constrain the design of other components? The network?

2. Send a message "ok_to_open?" Wait for responses.

   How many responses? How long to wait? What if a component has failed and never responds?

6

Deterministic Models are Useful

A model is deterministic if, given the initial state and the inputs, the model defines exactly one behavior.

Determinism

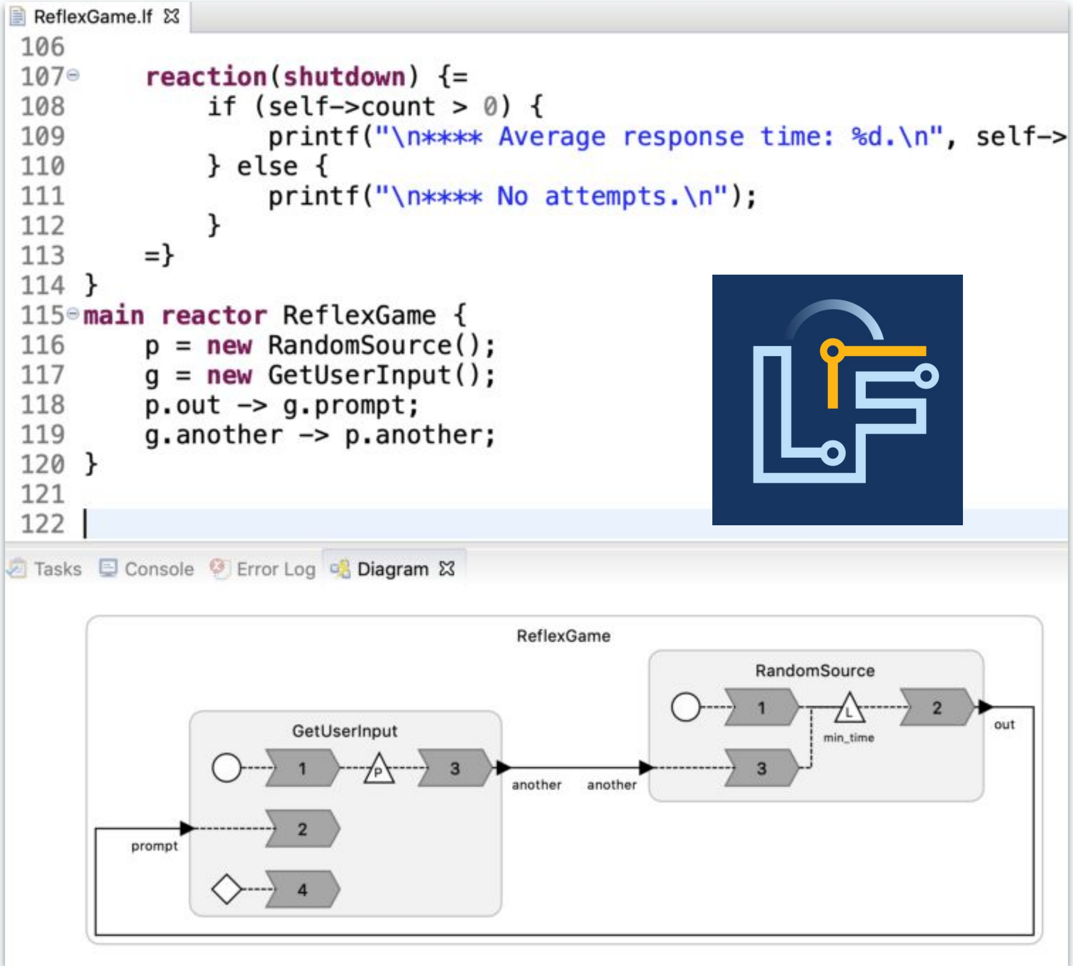
- ❖ Enables testing and more tractable analysis
- ❖ Makes simulation more useful
- ❖ Allows verification to scale better

Concurrency, Distribution are Necessary

- ❖ Performance, scalability, flexibility, complexity
  - ➢ Cyber-physical systems
- ❖ Dominant parallel and distributed programming paradigms have relinquished determinism: "everything is asynchronous"
  - ➢ Actors, publish-subscribe, service-oriented architectures, distributed shared memory
  - ➢ Even in safety-critical domains: e.g., ROS2, Autosar Adaptive Platform[1], etc.

[1] Menard, Christian, et al. "Achieving determinism in adaptive AUTOSAR." *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020.

Lingua Franca: It's About Time



```
  reaction(shutdown) {=
    if (self->count > 0) {
      printf("\n**** Average response time: %d.\n", self->
    } else {
      printf("\n**** No attempts.\n");
    }
  =}
}
main reactor ReflexGame {
  p = new RandomSource();
  g = new GetUserInput();
  p.out -> g.prompt;
  g.another -> p.another;
}
```

- ❖ Polyglot
- ❖ Explicit dependencies
- ❖ Discrete Event semantics
- ❖ Synchronous *reactions*
- ❖ *Actions* relate *logical time* or *physical time*

Logical Time and Physical time

Logical Time

- Steps or 'ticks'
- Discrete
- Absolute Simultaneity

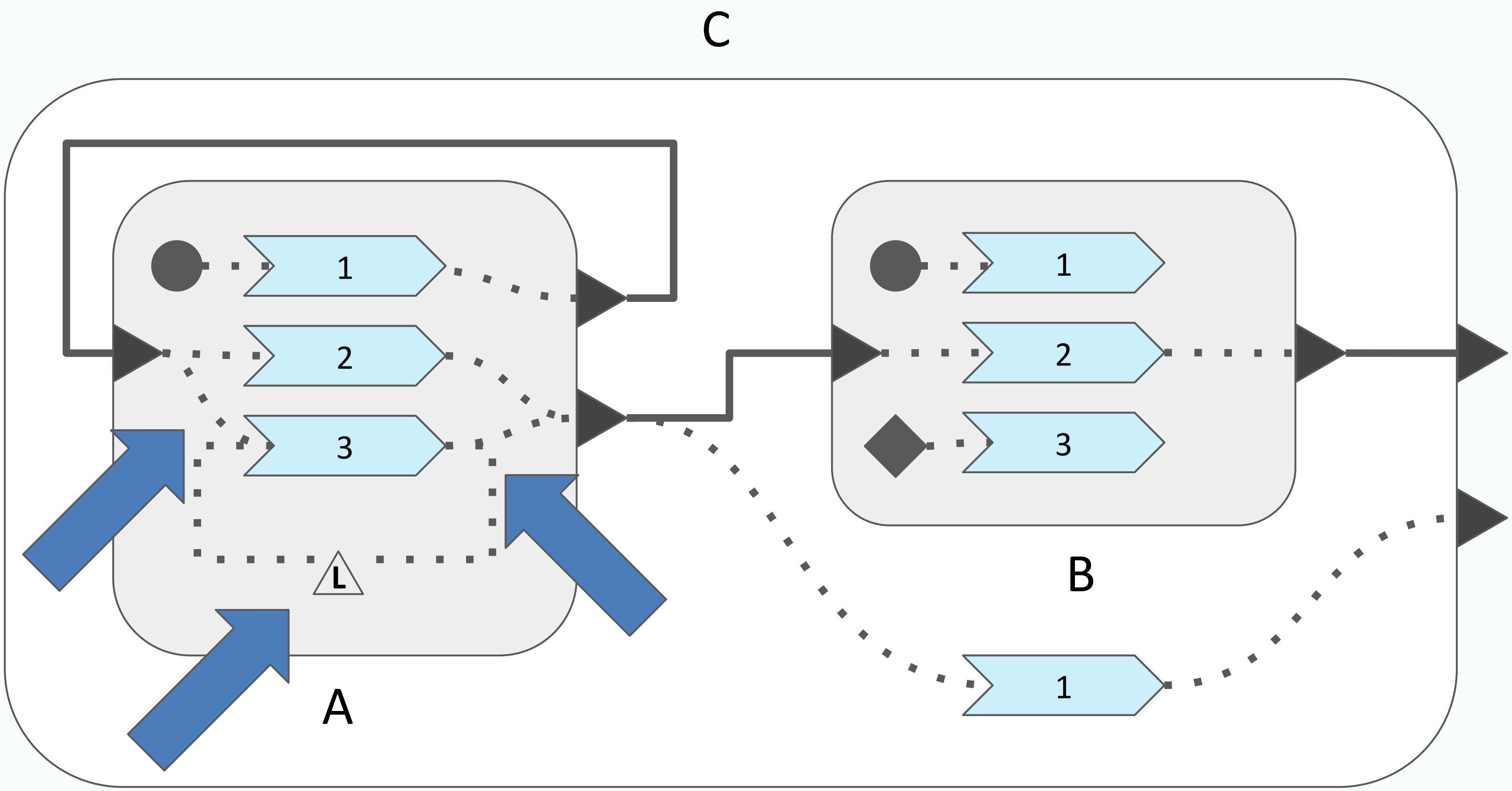
- Deadlines
- Federation
- Fault handling

Physical Time

- Measurements
- Continuous
- Relativistic Simultaneity

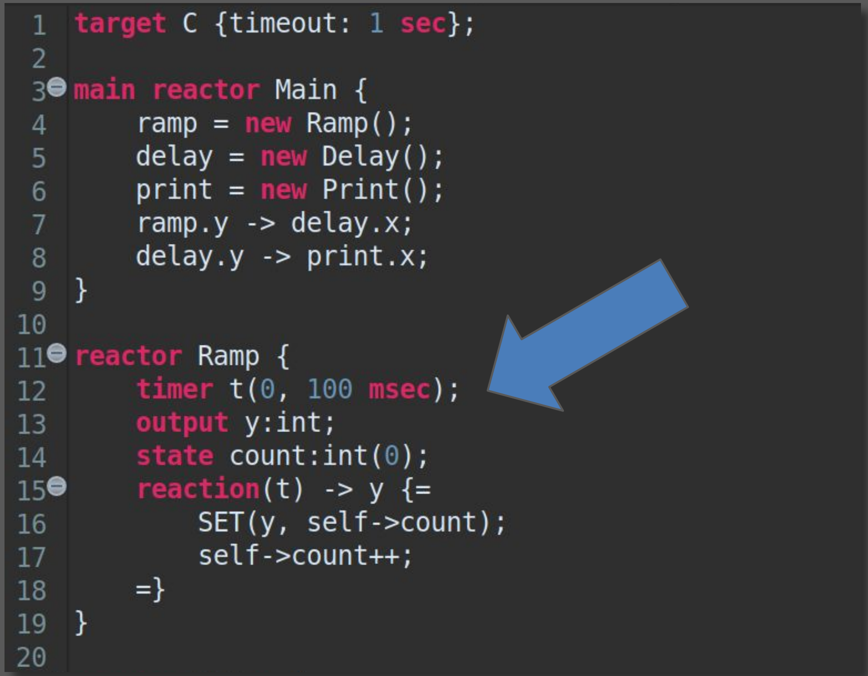
The Persistence of Memory — Salvador Dalí

Reactors[1] in a Nutshell

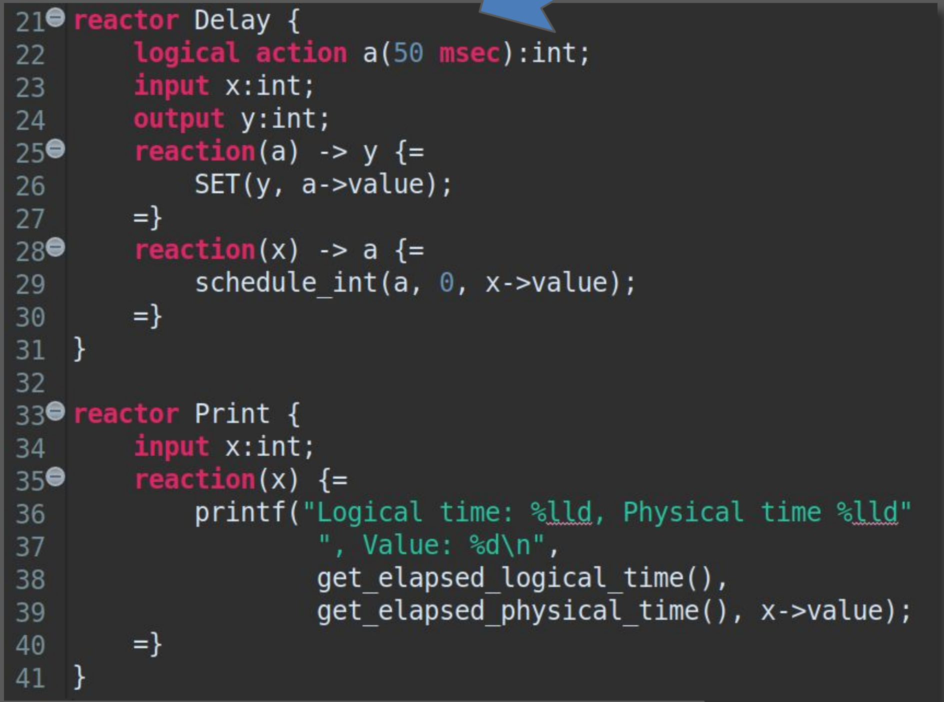


[1] Lohstroh, Marten, et al. "Reactors: A deterministic model for composable reactive systems." *Cyber Physical Systems. Model-Based Design*. Springer, Cham, 2019. 59-85.

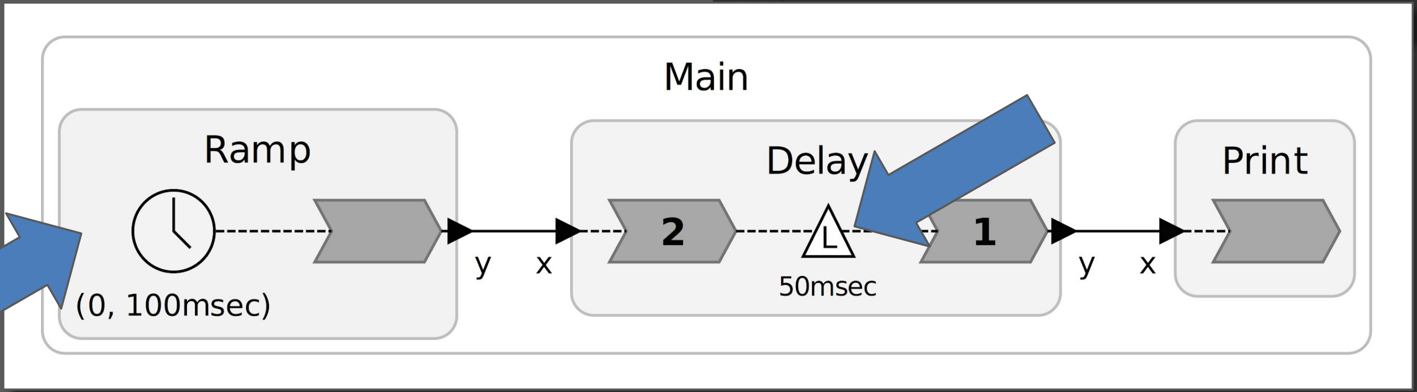
Logical Actions



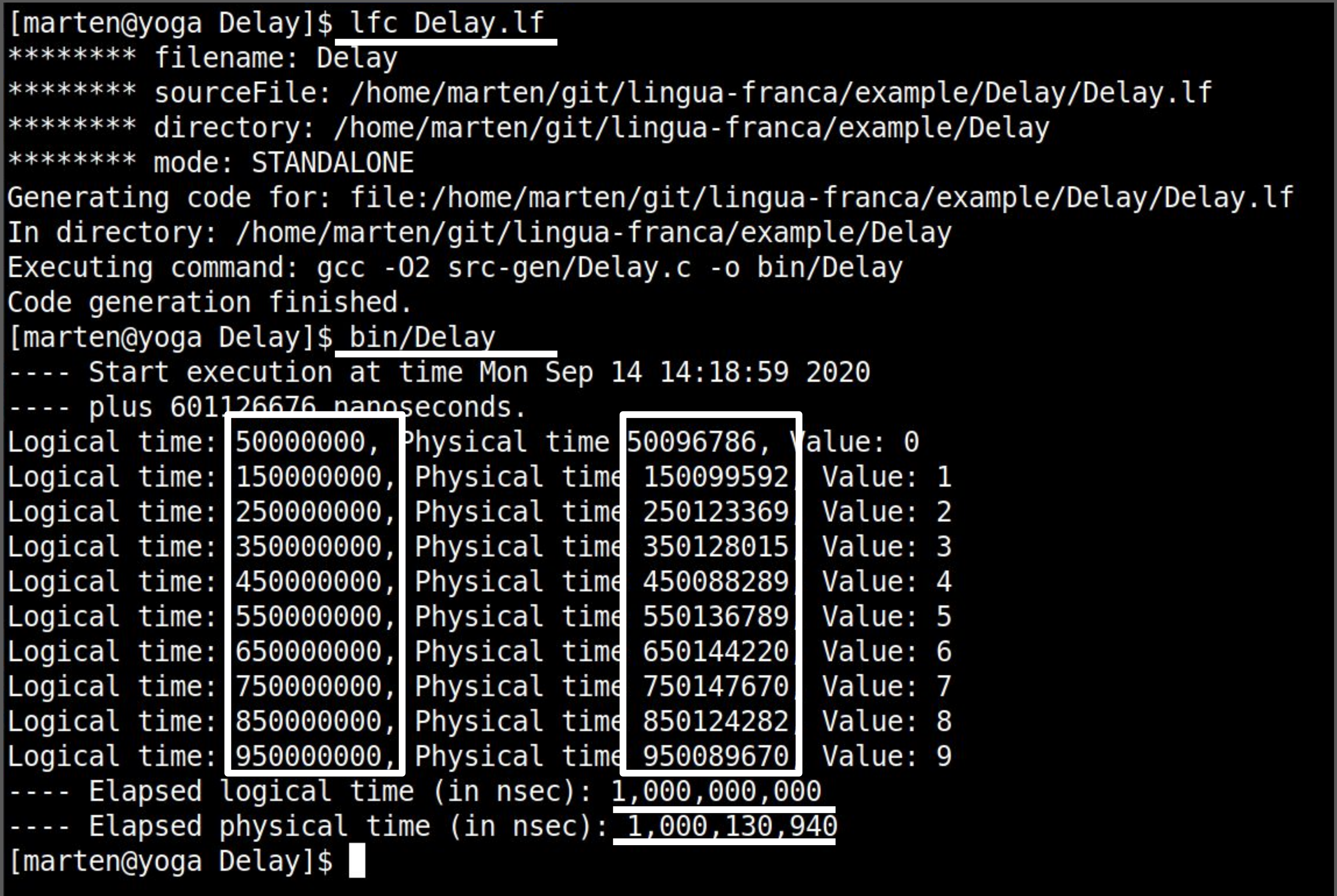
```
target C {timeout: 1 sec};

main reactor Main {
    ramp = new Ramp();
    delay = new Delay();
    print = new Print();
    ramp.y -> delay.x;
    delay.y -> print.x;
}

reactor Ramp {
    timer t(0, 100 msec);
    output y:int;
    state count:int(0);
    reaction(t) -> y {=
        SET(y, self->count);
        self->count++;
    =}
}

```



```
reactor Delay {
    logical action a(50 msec):int;
    input x:int;
    output y:int;
    reaction(a) -> y {=
        SET(y, a->value);
    =}
    reaction(x) -> a {=
        schedule_int(a, 0, x->value);
    =}
}

reactor Print {
    input x:int;
    reaction(x) {=
        printf("Logical time: %lld, Physical time %lld"
               ", Value: %d\n",
               get_elapsed_logical_time(),
               get_elapsed_physical_time(), x->value);
    =}
}
```

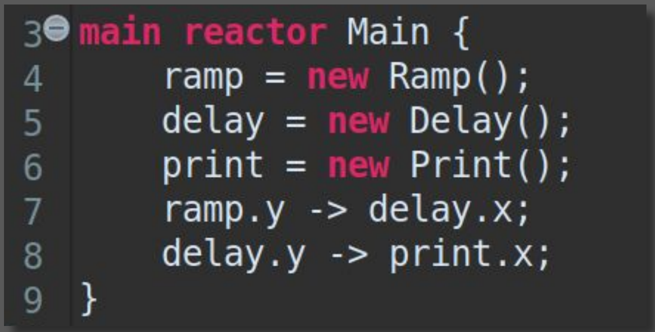


Logical Actions



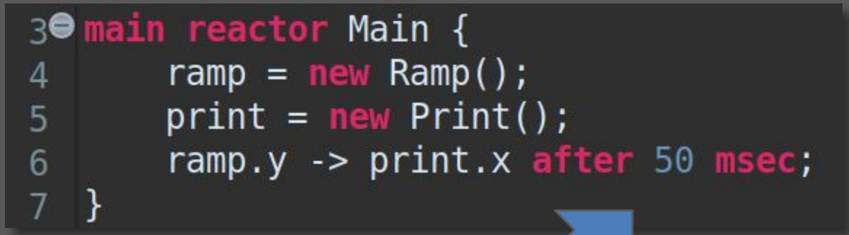
```
[marten@yoga Delay]$ lfc Delay.lf
******** filename: Delay
******** sourceFile: /home/marten/git/lingua-franca/example/Delay/Delay.lf
******** directory: /home/marten/git/lingua-franca/example/Delay
******** mode: STANDALONE
Generating code for: file:/home/marten/git/lingua-franca/example/Delay/Delay.lf
In directory: /home/marten/git/lingua-franca/example/Delay
Executing command: gcc -O2 src-gen/Delay.c -o bin/Delay
Code generation finished.
[marten@yoga Delay]$ bin/Delay
---- Start execution at time Mon Sep 14 14:18:59 2020
---- plus 601126676 nanoseconds.
Logical time: 50000000, Physical time 50096786, Value: 0
Logical time: 150000000, Physical time 150099592, Value: 1
Logical time: 250000000, Physical time 250123369, Value: 2
Logical time: 350000000, Physical time 350128015, Value: 3
Logical time: 450000000, Physical time 450088289, Value: 4
Logical time: 550000000, Physical time 550136789, Value: 5
Logical time: 650000000, Physical time 650144220, Value: 6
Logical time: 750000000, Physical time 750147670, Value: 7
Logical time: 850000000, Physical time 850124282, Value: 8
Logical time: 950000000, Physical time 950089670, Value: 9
---- Elapsed logical time (in nsec): 1,000,000,000
---- Elapsed physical time (in nsec): 1,000,130,940
[marten@yoga Delay]$ █
```

The **after** Keyword

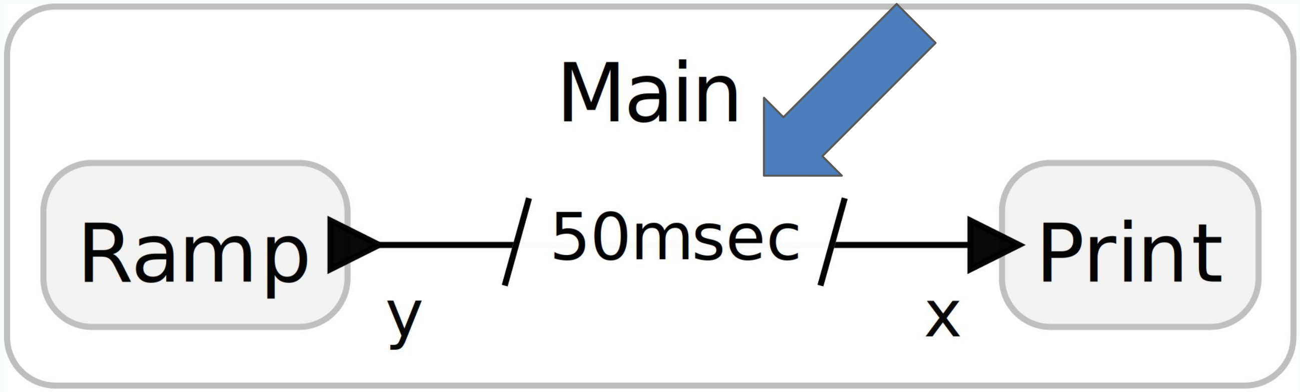


```
main reactor Main {
    ramp = new Ramp();
    delay = new Delay();
    print = new Print();
    ramp.y -> delay.x;
    delay.y -> print.x;
}
```

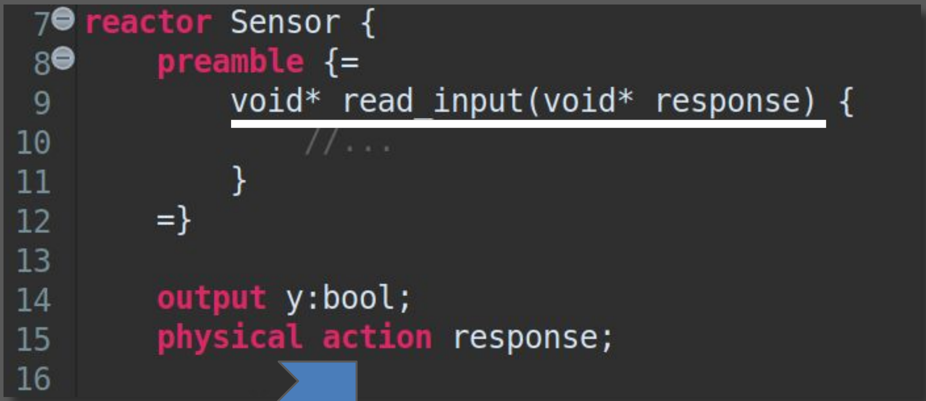
=



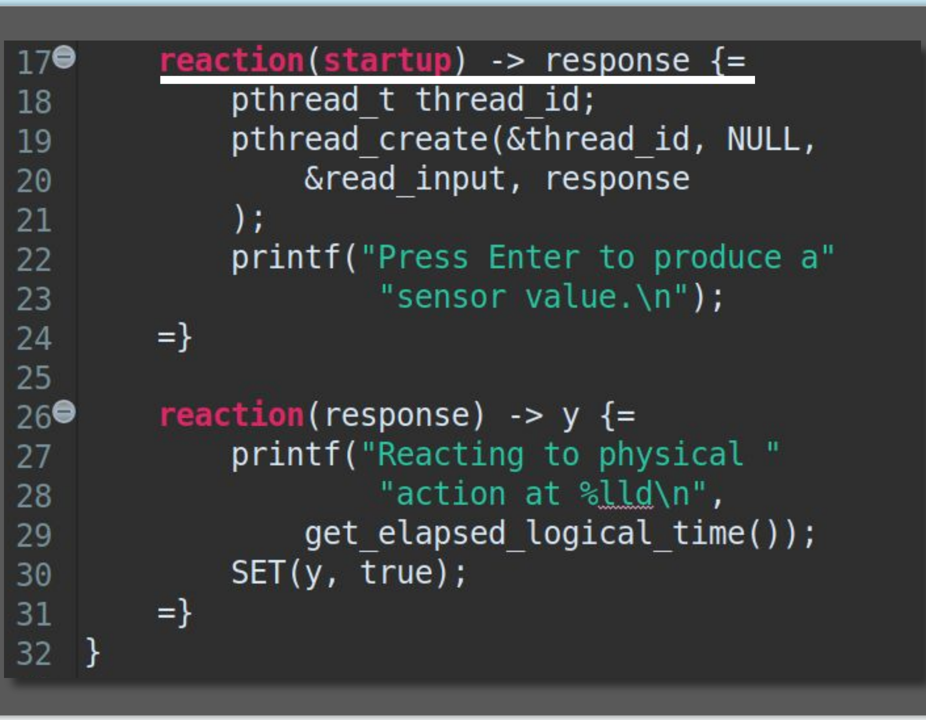
```
main reactor Main {
    ramp = new Ramp();
    print = new Print();
    ramp.y -> print.x after 50 msec;
}
```



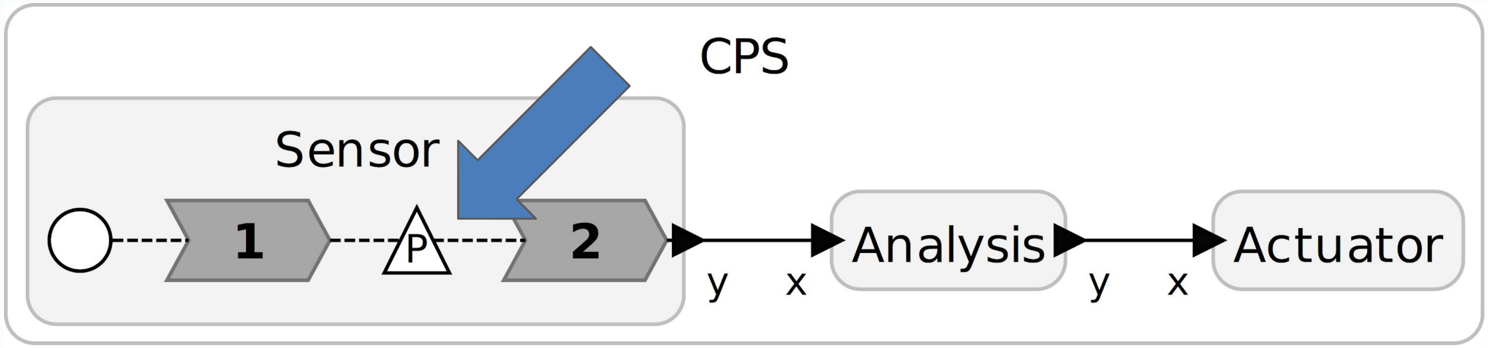
Physical Actions



```
reactor Sensor {
    preamble {=
        void* read_input(void* response) {
            //...
        }
    =}

    output y:bool;
    physical action response;

```



```
    reaction(startup) -> response {=
        pthread_t thread_id;
        pthread_create(&thread_id, NULL,
            &read_input, response
        );
        printf("Press Enter to produce a"
                "sensor value.\n");
    =}

    reaction(response) -> y {=
        printf("Reacting to physical "
                "action at %lld\n",
            get_elapsed_logical_time());
        SET(y, true);
    =}
}
```



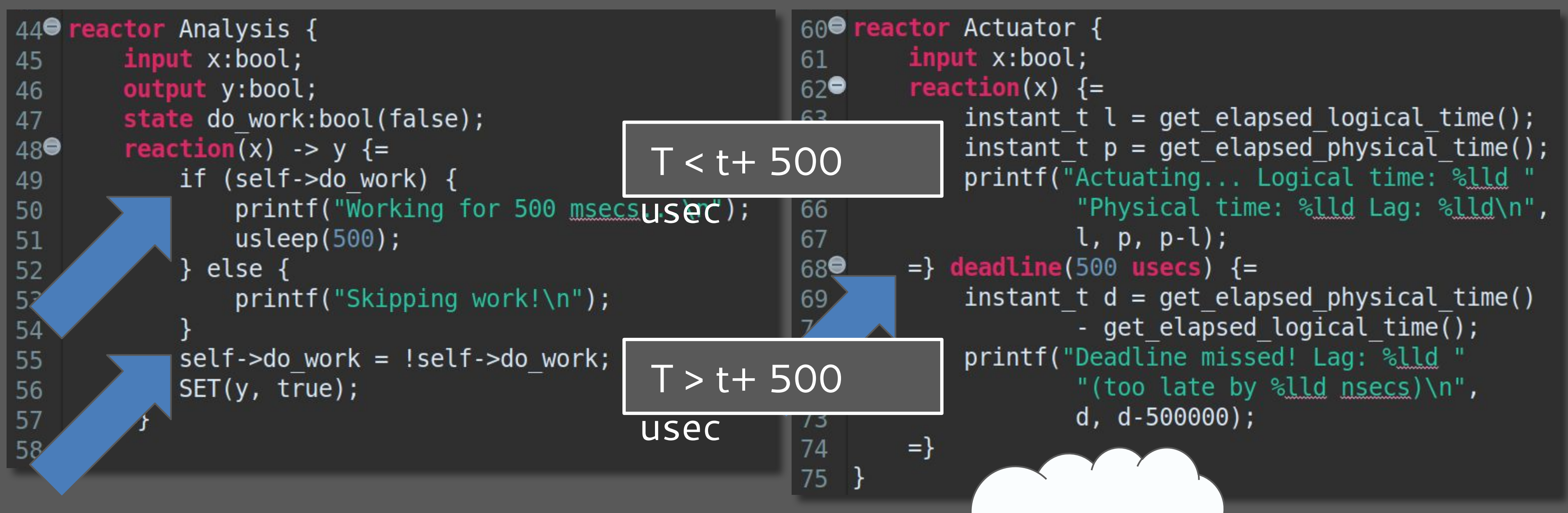
Physical Actions

Determinism

A model is deterministic if, given the initial state and the <u>inputs</u>, the model defines exactly one behavior.

- ❖ Tags assigned to events scheduled through a physical action are treated as *<u>inputs</u>*
- ❖ LF ensures that the logical time never gets ahead of physical time; further processing is exclusively determined by tags

Deadlines

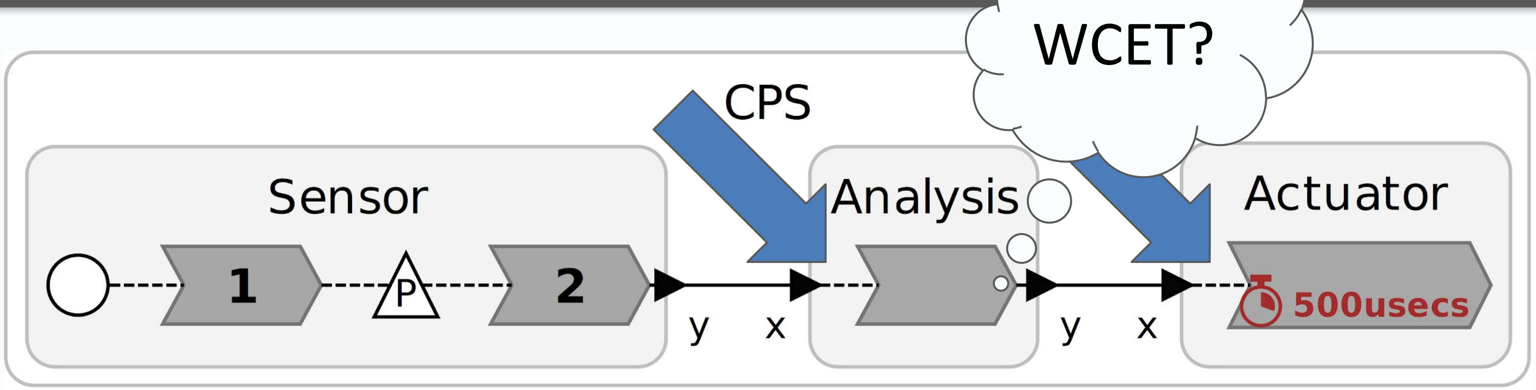


```
reactor Analysis {
    input x:bool;
    output y:bool;
    state do_work:bool(false);
    reaction(x) -> y {=
        if (self->do_work) {
            printf("Working for 500 msecs...\n");
            usleep(500);
        } else {
            printf("Skipping work!\n");
        }
        self->do_work = !self->do_work;
        SET(y, true);
    }
```

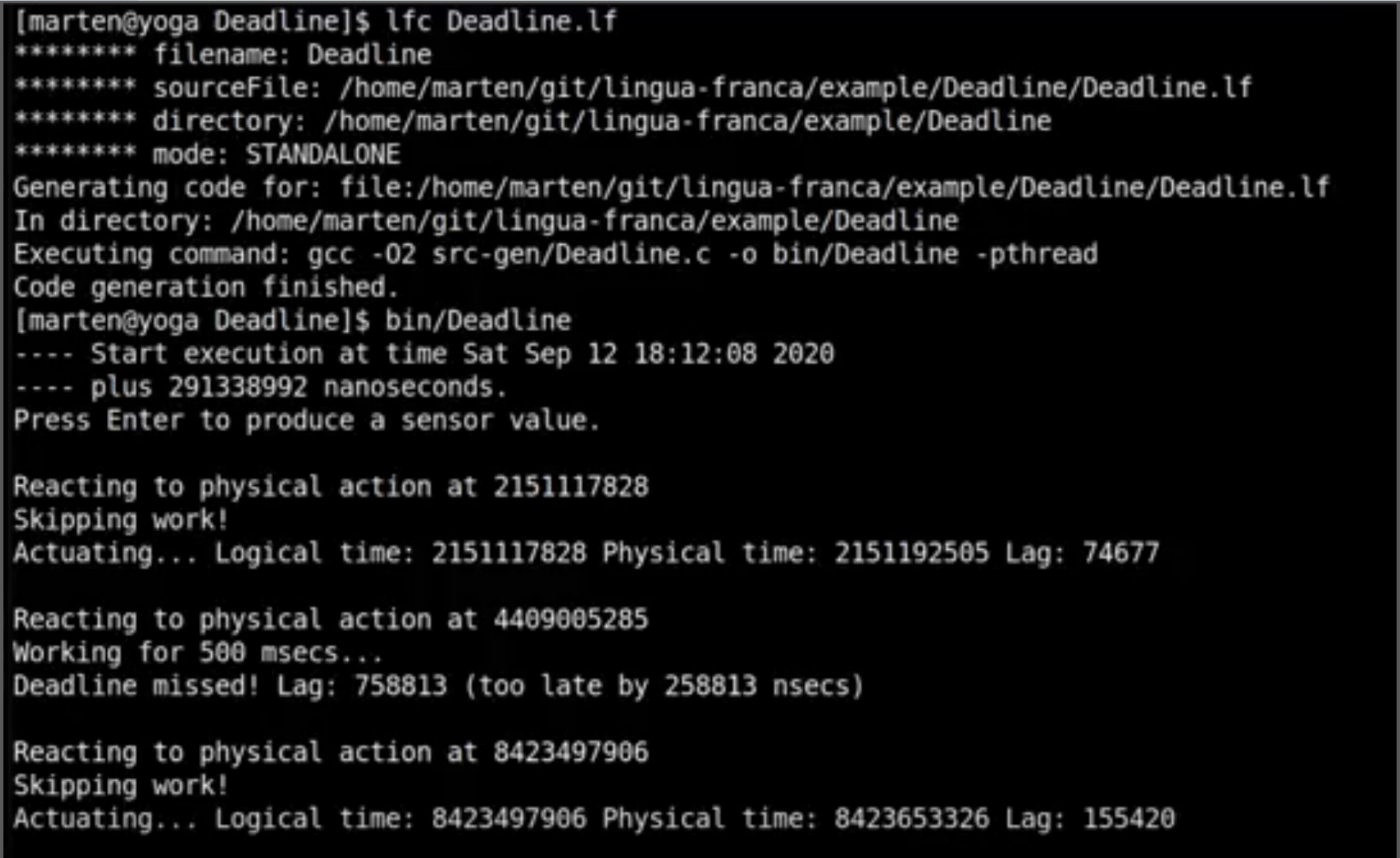
T < t+ 500 usec

T > t+ 500 usec

```
reactor Actuator {
    input x:bool;
    reaction(x) {=
        instant_t l = get_elapsed_logical_time();
        instant_t p = get_elapsed_physical_time();
        printf("Actuating... Logical time: %lld "
            "Physical time: %lld Lag: %lld\n",
            l, p, p-l);
    =} deadline(500 usecs) {=
        instant_t d = get_elapsed_physical_time()
            - get_elapsed_logical_time();
        printf("Deadline missed! Lag: %lld "
            "(too late by %lld nsecs)\n",
            d, d-500000);
    =}
}
```



Deadlines



```
[marten@yoga Deadline]$ lfc Deadline.lf
******** filename: Deadline
******** sourceFile: /home/marten/git/lingua-franca/example/Deadline/Deadline.lf
******** directory: /home/marten/git/lingua-franca/example/Deadline
******** mode: STANDALONE
Generating code for: file:/home/marten/git/lingua-franca/example/Deadline/Deadline.lf
In directory: /home/marten/git/lingua-franca/example/Deadline
Executing command: gcc -O2 src-gen/Deadline.c -o bin/Deadline -pthread
Code generation finished.
[marten@yoga Deadline]$ bin/Deadline
---- Start execution at time Sat Sep 12 18:12:08 2020
---- plus 291338992 nanoseconds.
Press Enter to produce a sensor value.

Reacting to physical action at 2151117828
Skipping work!
Actuating... Logical time: 2151117828 Physical time: 2151192505 Lag: 74677

Reacting to physical action at 4409005285
Working for 500 msecs...
Deadline missed! Lag: 758813 (too late by 258813 nsecs)

Reacting to physical action at 8423497906
Skipping work!
Actuating... Logical time: 8423497906 Physical time: 8423653326 Lag: 155420
```

Deadlines

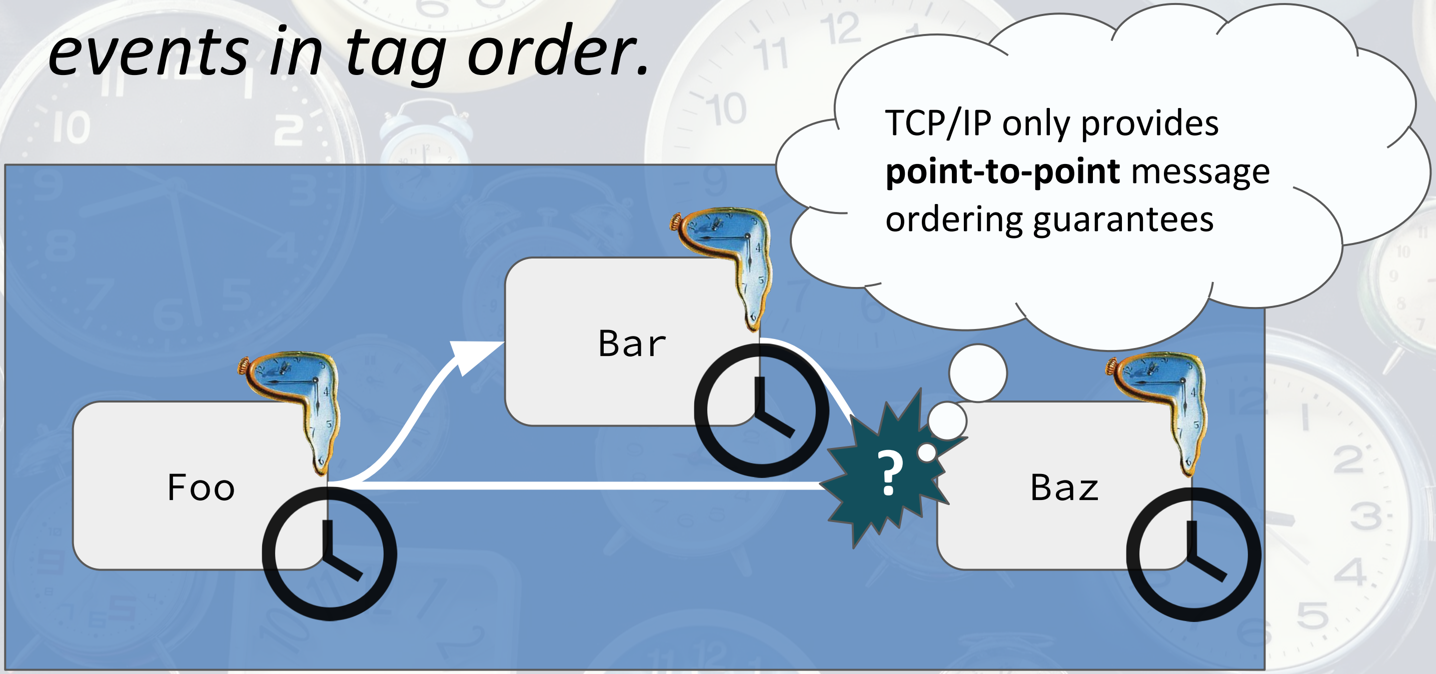
Determinism

A model is deterministic if, given the initial state and the inputs, the model defines exactly one behavior.

- ❖ Deadlines admit nondeterminism; the program is only deterministic if no deadlines are violated
- ❖ Dependent on factors outside the semantics of LF; deadline reactions are *fault handlers*

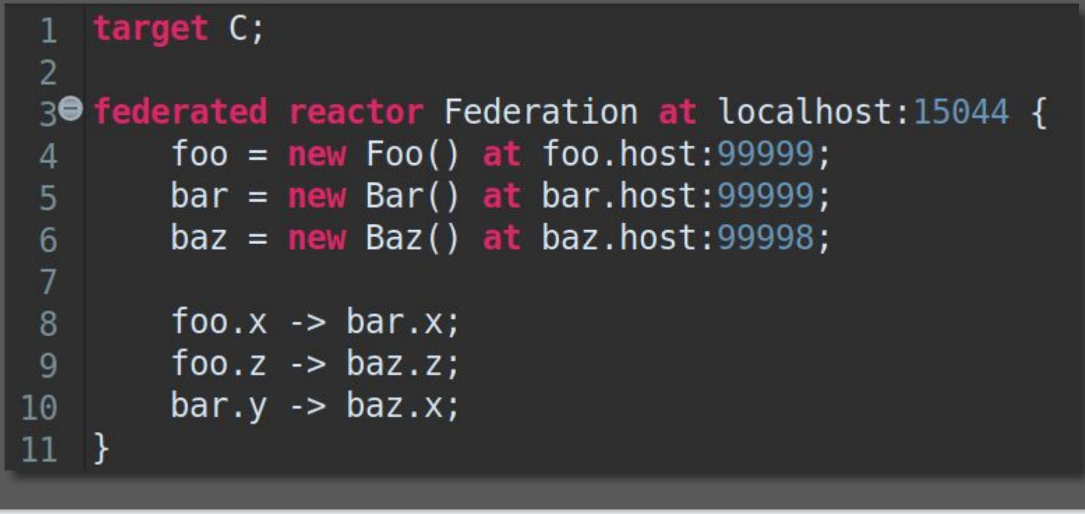
Federation: A Multiplicity of Timelines

***Goal**: have each federate observe all tagged events in tag order.*

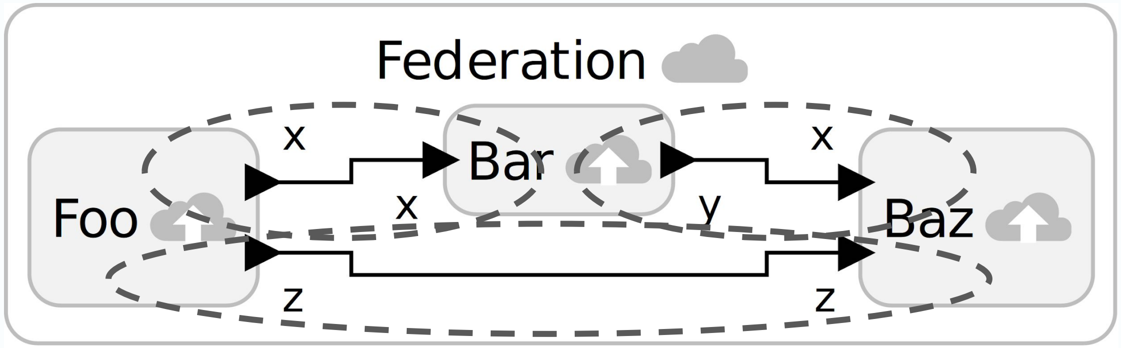


TCP/IP only provides **point-to-point** message ordering guarantees
Bar
Foo
?
Baz

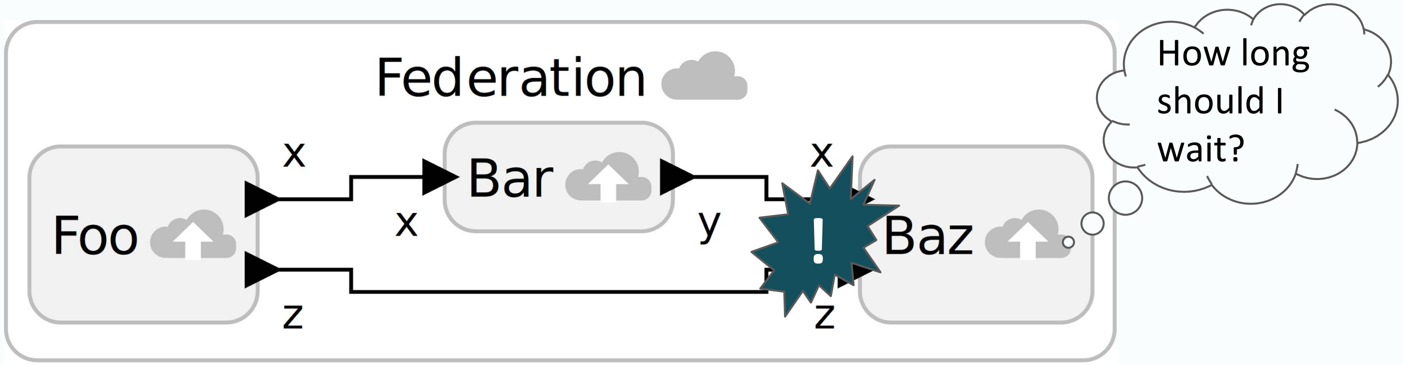
Federated LF Programs

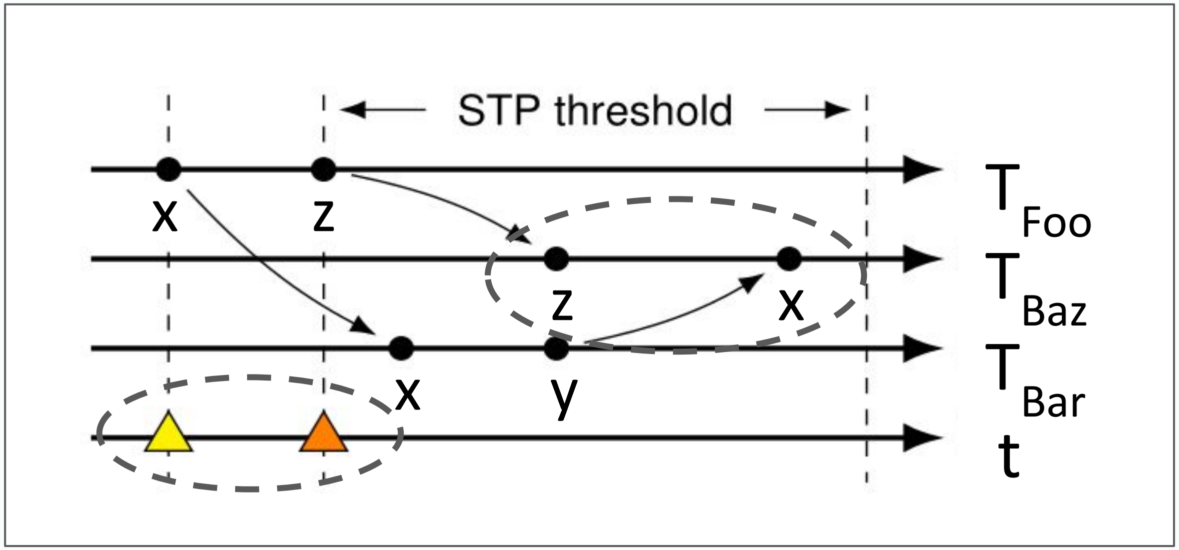


```
target C;

federated reactor Federation at localhost:15044 {
    foo = new Foo() at foo.host:99999;
    bar = new Bar() at bar.host:99999;
    baz = new Baz() at baz.host:99998;

    foo.x -> bar.x;
    foo.z -> baz.z;
    bar.y -> baz.x;
}
```



PTIDES[1] Safe-To-Process Analysis





STP Requires Bounds on:

- ➢ Execution times
- ➢ Network latency
- ➢ Clock synchronization error

[1] Zhao, Yang, Jie Liu, and Edward A. Lee. "A programming model for time-synchronized distributed real-time systems." *RTAS '07*. IEEE, 2007.

Assumptions ⇒ Determinism

- Deadlines
  - WCET
  - Schedulability
- Federated Execution
  - Execution times
  - Network latency
  - Clock synchronization error
  - Logical time delays
  - Deadlines

Related Work

- ❖ Modeling
  - ➢ MARTE/CCSL (Mallet et al.)
  - ➢ TESL (Boulanger et al.)
- ❖ Synchronous Languages
  - ➢ SIGNAL (Le Guernic, Benveniste, Gautier)
  - ➢ Multiclock Esterel (Berry and Sentovich)
- ❖ @ FDL '20
  - ➢ Timed C (Natarajan and Broman)
  - ➢ Sparse Synchronous Model (Edwards and Hui)

Conclusion

- Determinism can be achieved under assumptions on the **relation between logical time and physical time**
- Lingua Franca lets the programmer make these **assumptions explicit**
- **Violations** of these assumptions are **detectable at runtime**

Acknowledgements

The core Lingua Franca software development team current consists of: Soroush Bateni, Edward A. Lee, Shaokai Lin, Marten Lohstroh, Christian Menard, Alexander Schulz-Rosengarten, and Matt Weber.

Others who have influenced LF with their ideas are: Janette Cardoso, Jeronimo Castrillon, Patricia Derler, Christopher Gill, Andrés Goens, Íñigo Íncer Romeo, Alberto Sangiovanni-Vincentelli, Martin Schoeberl, Sanjit Seshia, and Marjan Sirjani.

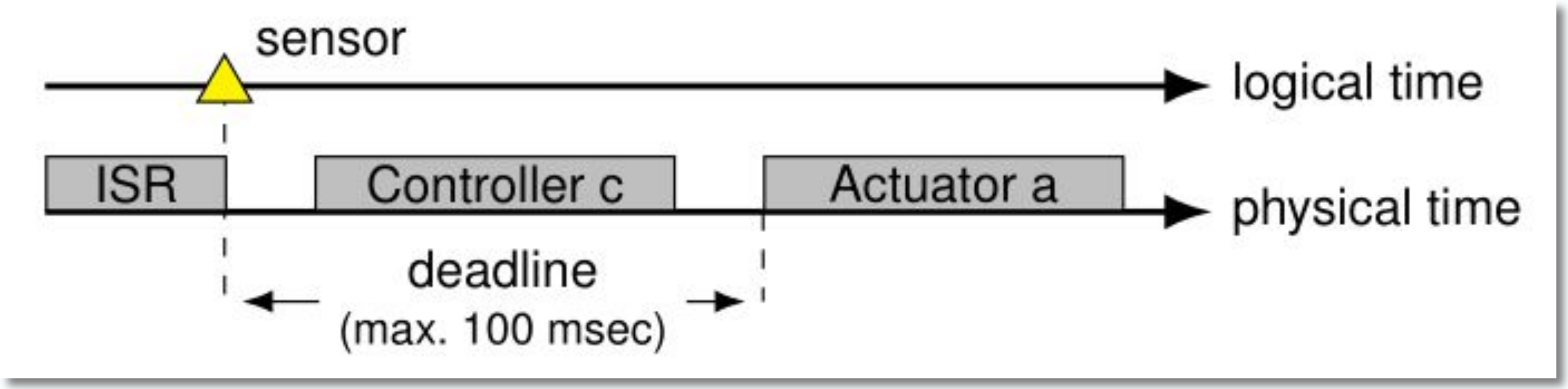
The work in this paper was supported in part by the National Science Foundation (NSF), award #CNS-1836601 (Reconciling Safety with the Internet) and the iCyPhy (Industrial Cyber-Physical Systems) research center, supported by Denso, Siemens, and Toyota.

Visit our GitHub:

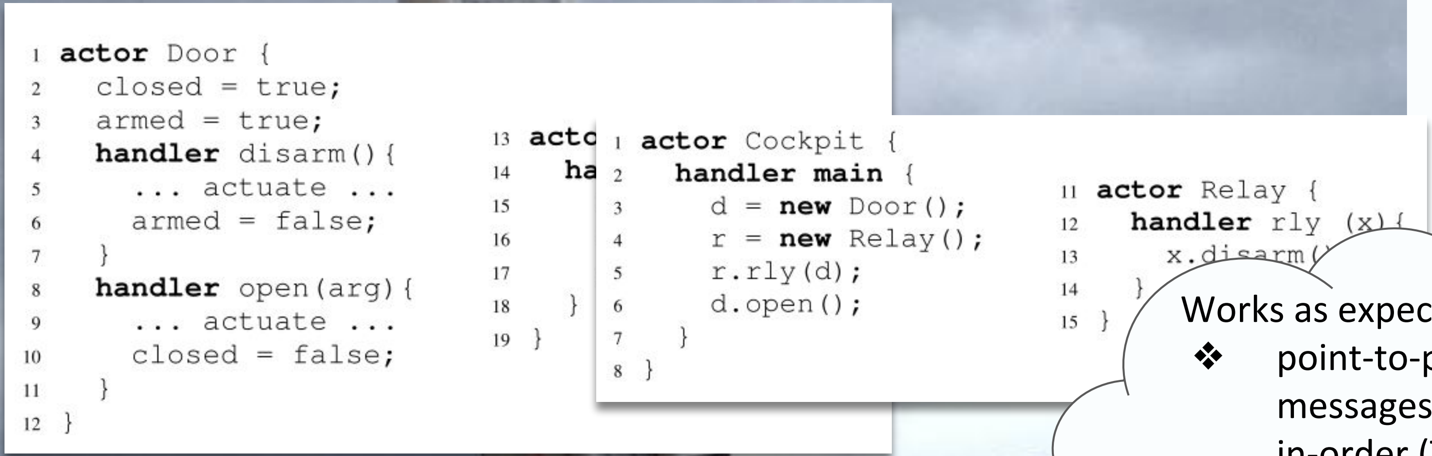
LINGUA
FRANCA

github.com/icyphy/lingua-franca

Deadlines


sensor
logical time
ISR
Controller c
Actuator a
physical time
deadline
(max. 100 msec)

A Cyber-Physical Example



```
actor Door {
  closed = true;
  armed = true;
  handler disarm(){
    ... actuate ...
    armed = false;
  }
  handler open(arg){
    ... actuate ...
    closed = false;
  }
}
```

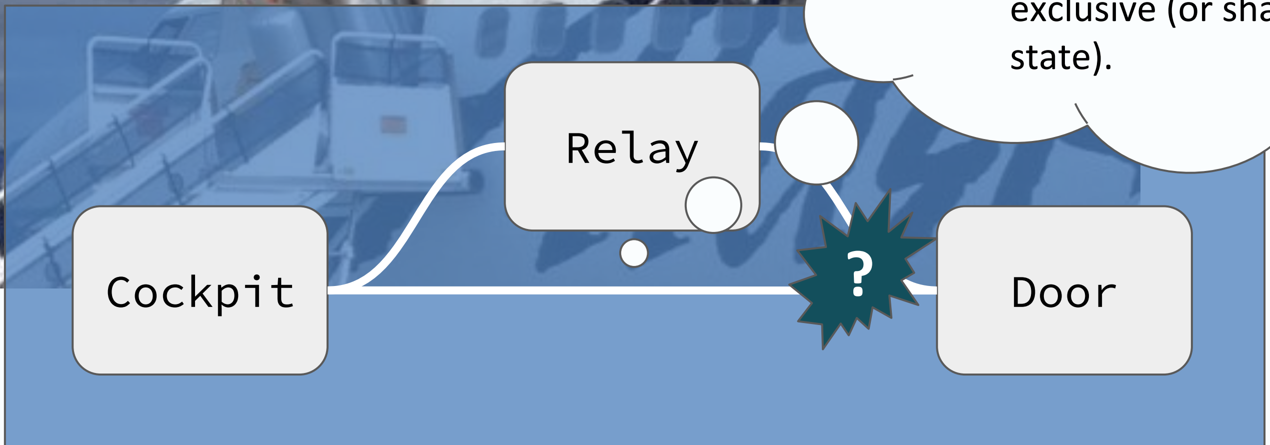
```
acto
  ha



    }
}
```

```
actor Cockpit {
  handler main {
    d = new Door();
    r = new Relay();
    r.rly(d);
    d.open();
  }
}
```

```
actor Relay {
  handler rly (x){
    x.disarm(
  }
}
```

Works as expected if:

- ❖ point-to-point messages are delivered in-order (TCP/IP); and
- ❖ handlers are mutually exclusive (or share no state).


Relay
Cockpit
?
Door