Software Design for Cyber-Physical Systems

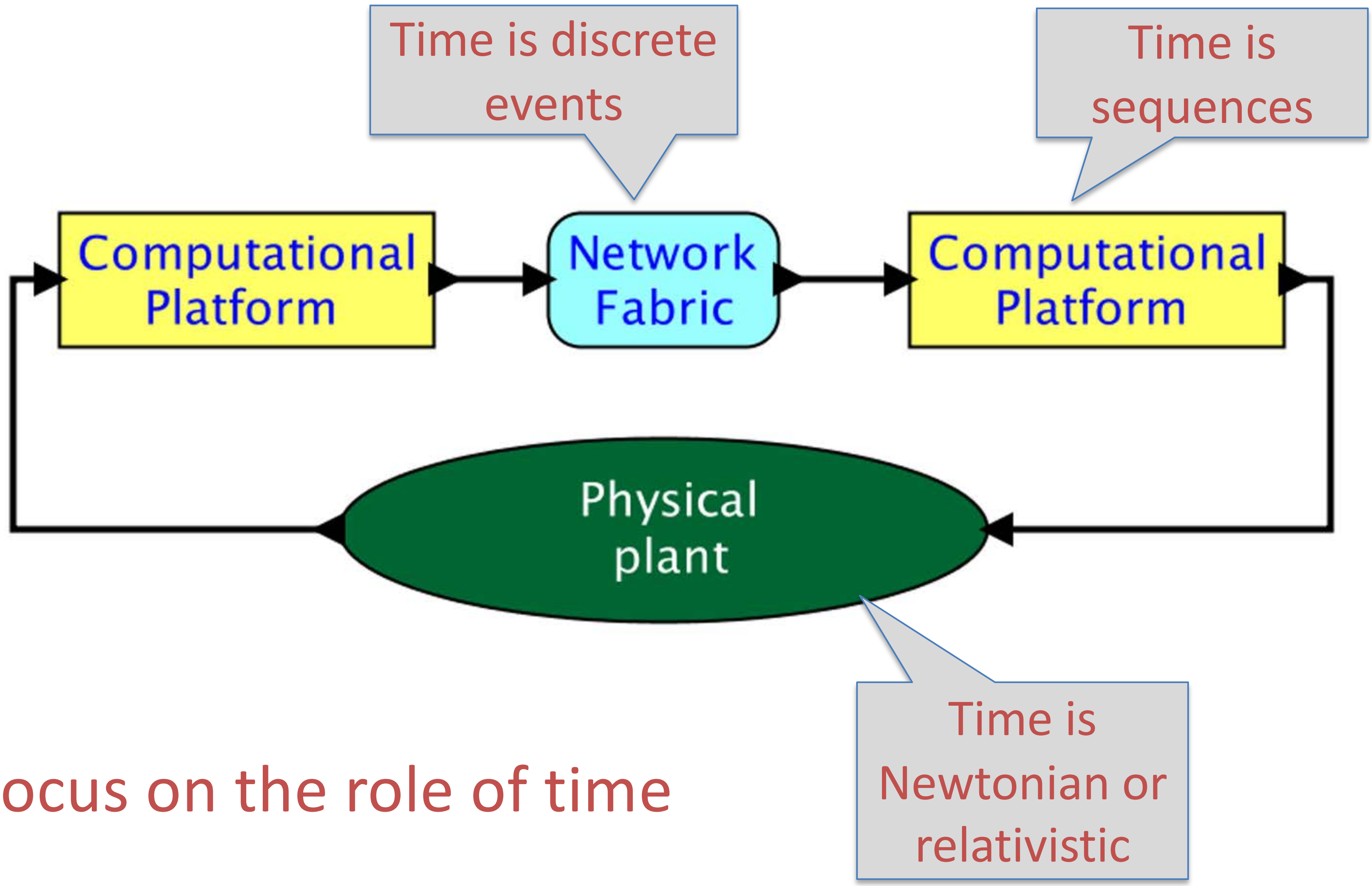
Edward A. Lee

Module 4: Time in Lingua Franca

Technical University of Vienna
Vienna, Austria, May 2022

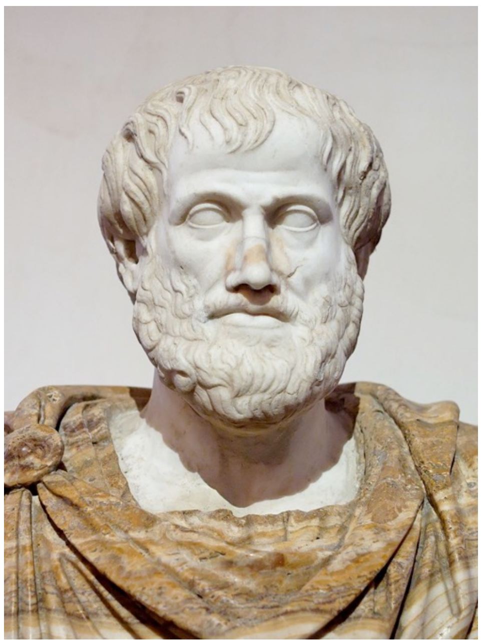
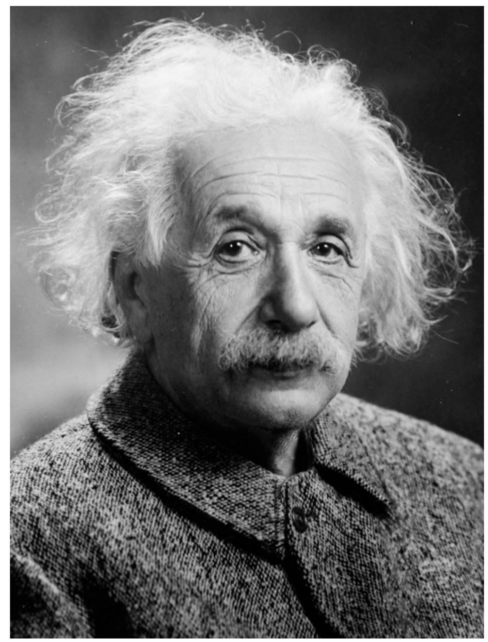
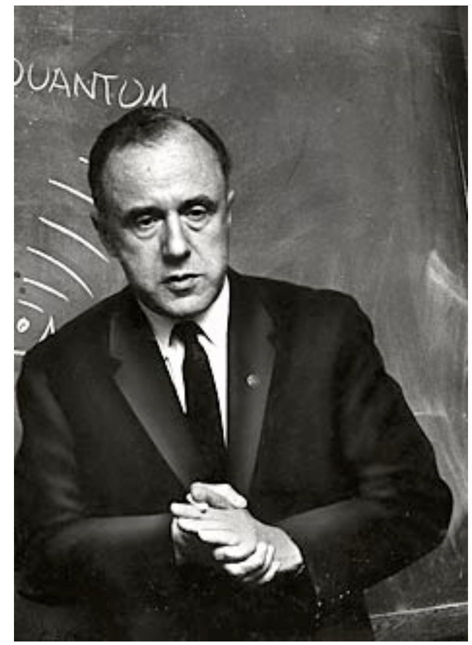
University of California, Berkeley

Cyber-Physical Systems: A Huge Modeling Challenge

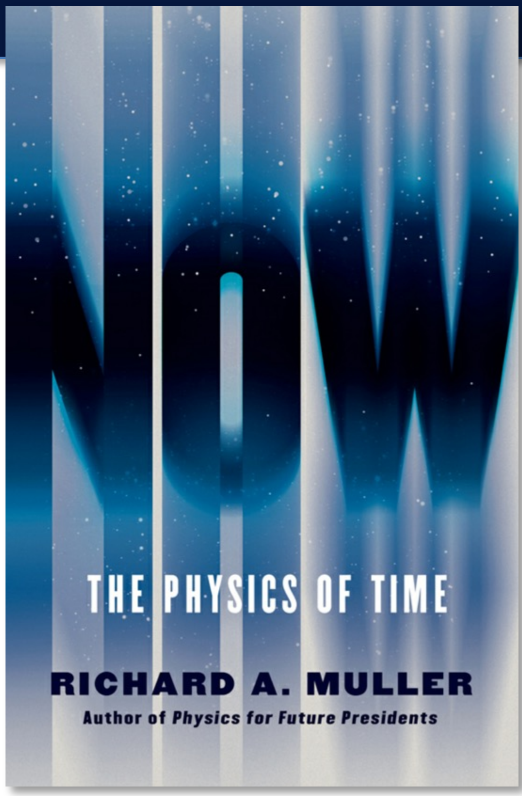


Focus on the role of time

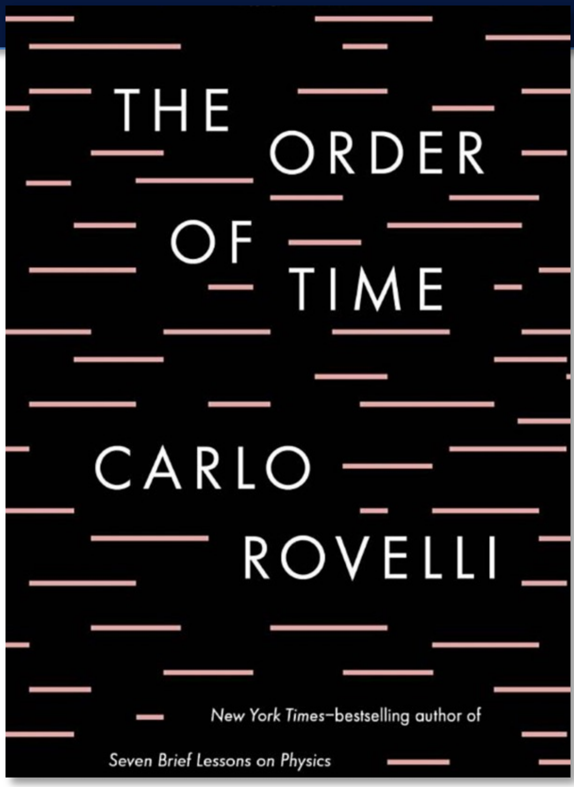
What is Time?

Change	Smooth	Relative	Discrete
			
Aristotle	Newton	Einstein	Wheeler

What is Time?



2016



2018

Muller: Gives a theory of time that requires big black holes to collide somewhere near us to test it.

Rovelli: “The nature of time is perhaps the greatest remaining mystery.”

How can we build systems based on something we do not understand

?

Deterministic Timing

Is our goal for our models to accurately reflect the timing of our implementation?

or

Is our goal for the implementation to accurately reflect the timing of the model?

If it's the latter, then deterministic timing makes perfect sense!

Desirable Properties in a Model of Time

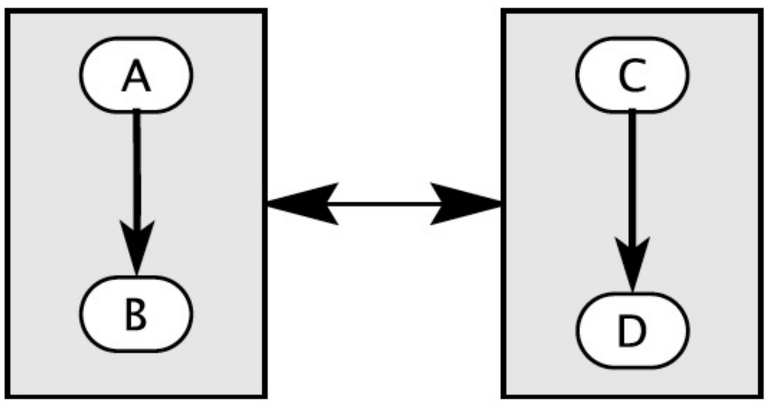
- A "**present**" that separates the past and future
- Support for **causality**
- A well-defined "**observer**"
- A notion of "**simultaneity**"

All are problematic in physics but useful in models.

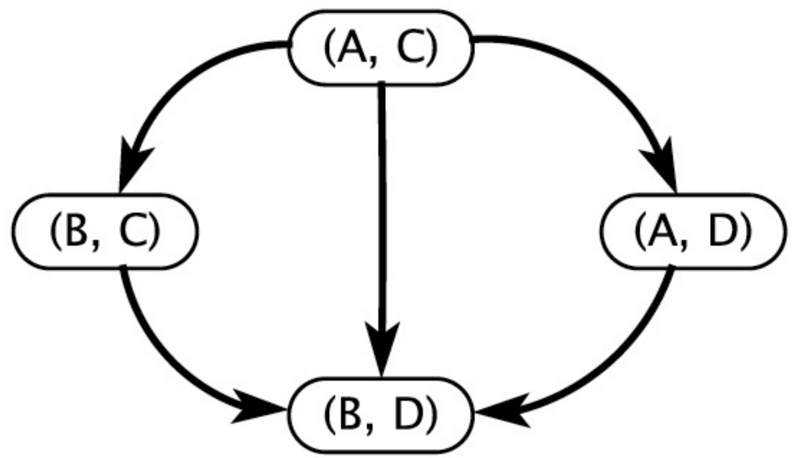
Order of Events and System State

There is no ground truth on the order in which events occur.

Einstein's train: https://youtu.be/wteiuxyqtoM

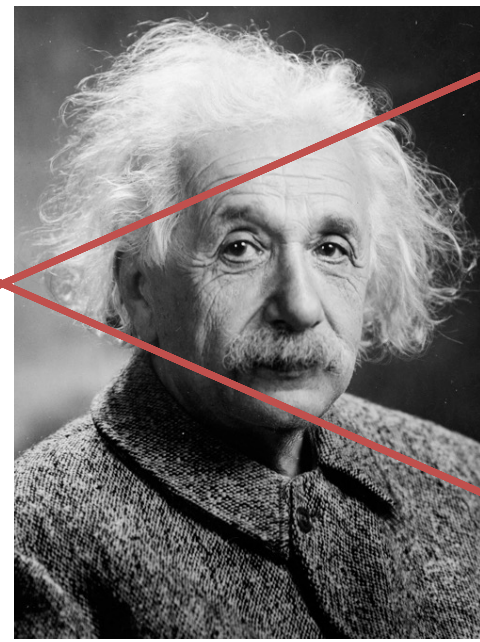
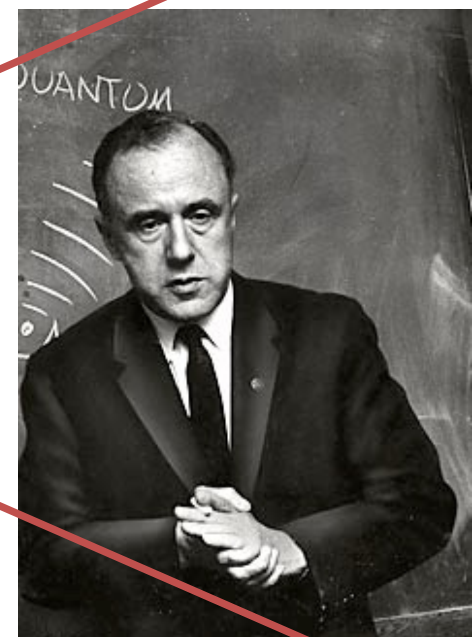


Physically separated state machines.



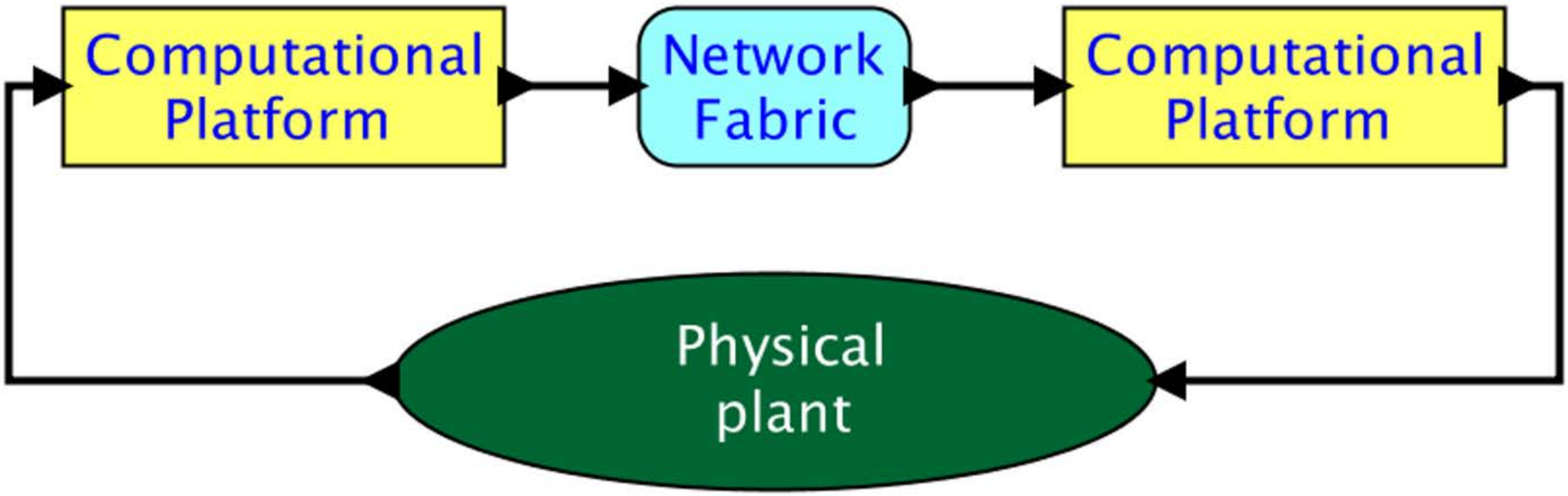
Transition system model.

What is Time?

Change	Smooth	Relative	Discrete
			
Aristotle	Newton	Einstein	Wheeler

All of these are about scientific models, not engineering models.

Cyber-Physical Systems: A Huge Modeling Challenge



The goal is for the implementation to accurately reflect the timing of the model.

So what should the model be?

Logical Time

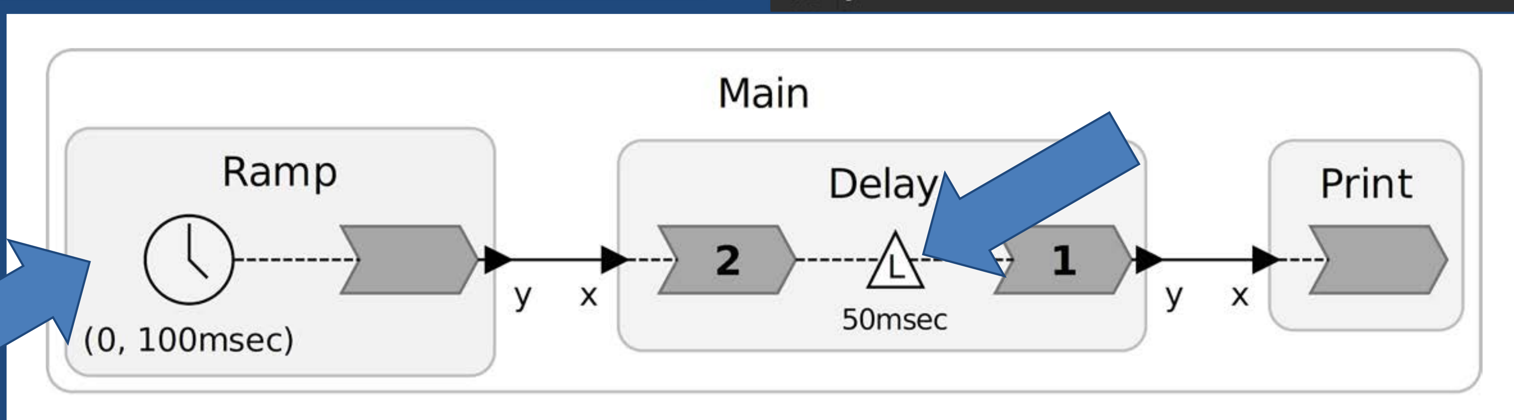
```
target C {timeout: 1 sec};

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 and Physical Time

```
[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]$ 
```

Time and Timers

From https://lf-lang.org:

LINGUA FRANCA

Download Docs Handbook Community

Target language: C

- Resources
- Writing Reactors
- A First Reactor
- Inputs and Outputs
- Parameters and State Variables
- Time and Timers
- Composing Reactors
- Reactions and Methods
- Causality Loops
- Extending Reactors
- Actions
- Superdense Time
- Deadlines

Time and Timers

This page is showing examples in the target language C. You can change the target language in the left sidebar.

Logical Time

A key property of Lingua Franca is **logical time**. All events occur at an instant in logical time. By default, the runtime system does its best to align logical time with **physical time**, which is some measurement of time on the execution platform. The **lag** is defined to be physical time minus logical time, and the goal of the runtime system is maintain a small non-negative lag.

The **lag** is allowed to go negative only if the `fast` target property or the —fast is set to `true`. In that case, the program will execute as fast as possible with no regard to physical time.

In Lingua Franca, **time** is a data type. A parameter, state variable, port, or action may have type **time**. In the C target, time values internally have type `instant_t` or `interval_t`, both of which are (usually) equivalent to the C type `long long`.

On this page

- Logical Time
- Time Values
- Timers
- Elapsed Time
- Comparing Logical and Physic...
- Simultaneity and Instantaneity
- Timeout
- Startup and Shutdown

Is this page helpful?

Yes No

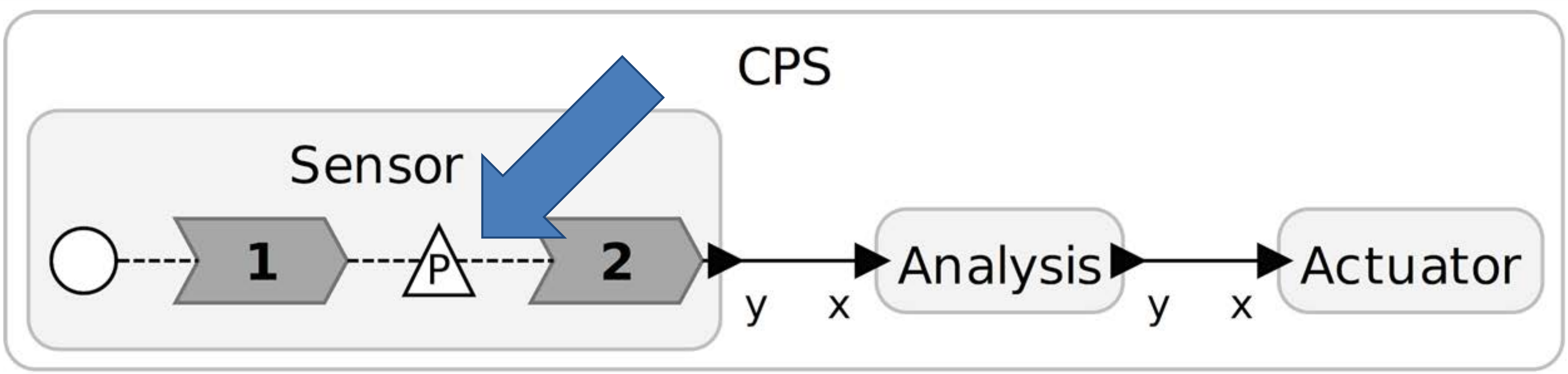
Asynchronous External Events

```
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);
    =}
}
```



Logical and Physical Actions

Target language: C

Resources

Writing Reactors

A First Reactor

Inputs and Outputs

Parameters and State Variables

Time and Timers

Composing Reactors

Reactions and Methods

Causality Loops

Extending Reactors

Actions

Superdense Time

Deadlines

Multiports and Banks

Preambles and Methods

Distributed Execution

Termination

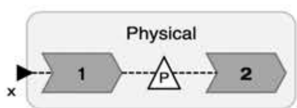
Tools

Reference

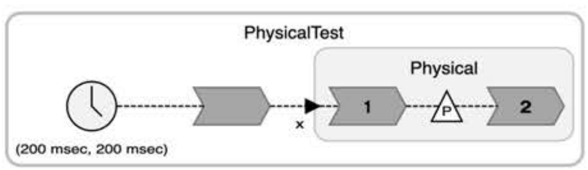
Physical Actions

A **physical action** is used to schedule reactions at logical times determined by the local physical clock. If a physical action with delay *d* is scheduled at *physical* time *T*, then the *logical time* assigned to the event is *T* + *d*. For example, the following reactor schedules the physical action `p` to trigger at a **logical time** equal to the **physical time** at which the input `x` arrives:

```
target C;
reactor Physical {
    input x:int;
    physical action a;
    reaction(x) -> a {=
        lf_schedule(a, 0);
    =}
    reaction(a) {=
        interval_t elapsed_time = lf_time_logical_elapsed();
        printf("Action triggered at logical time %lld nsec after start.\n", elapsed_time);
    =}
}
```



If you drive this with a timer, using for example the following structure:



then running the program will yield an output something like this:

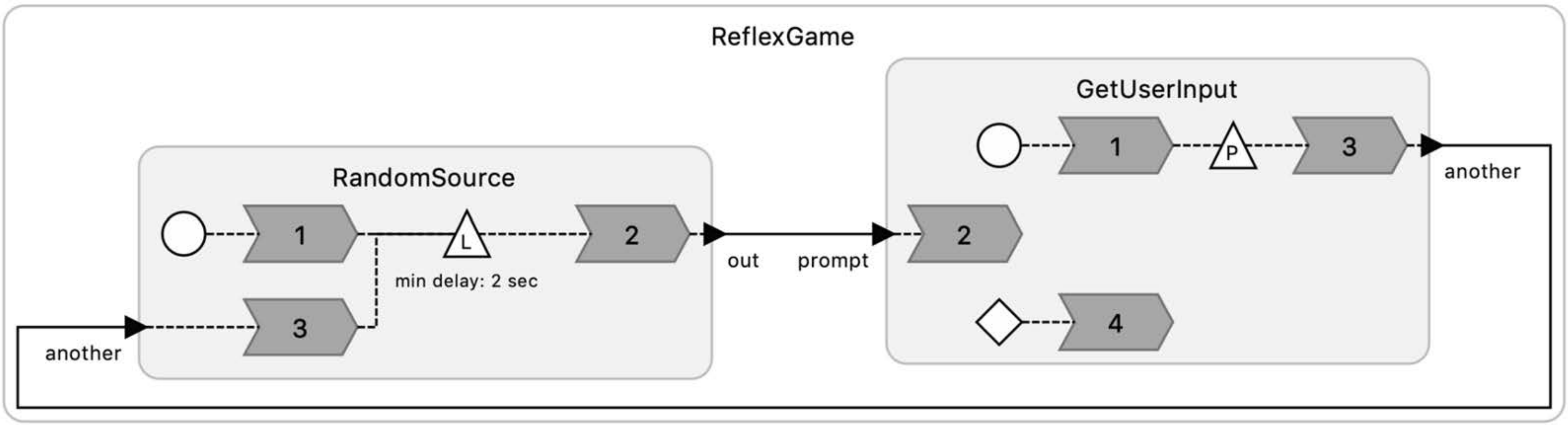
```
Action triggered at logical time 201491000 nsec after start.
Action triggered at logical time 403685000 nsec after start.
Action triggered at logical time 603669000 nsec after start.
...
```


Is this page helpful?

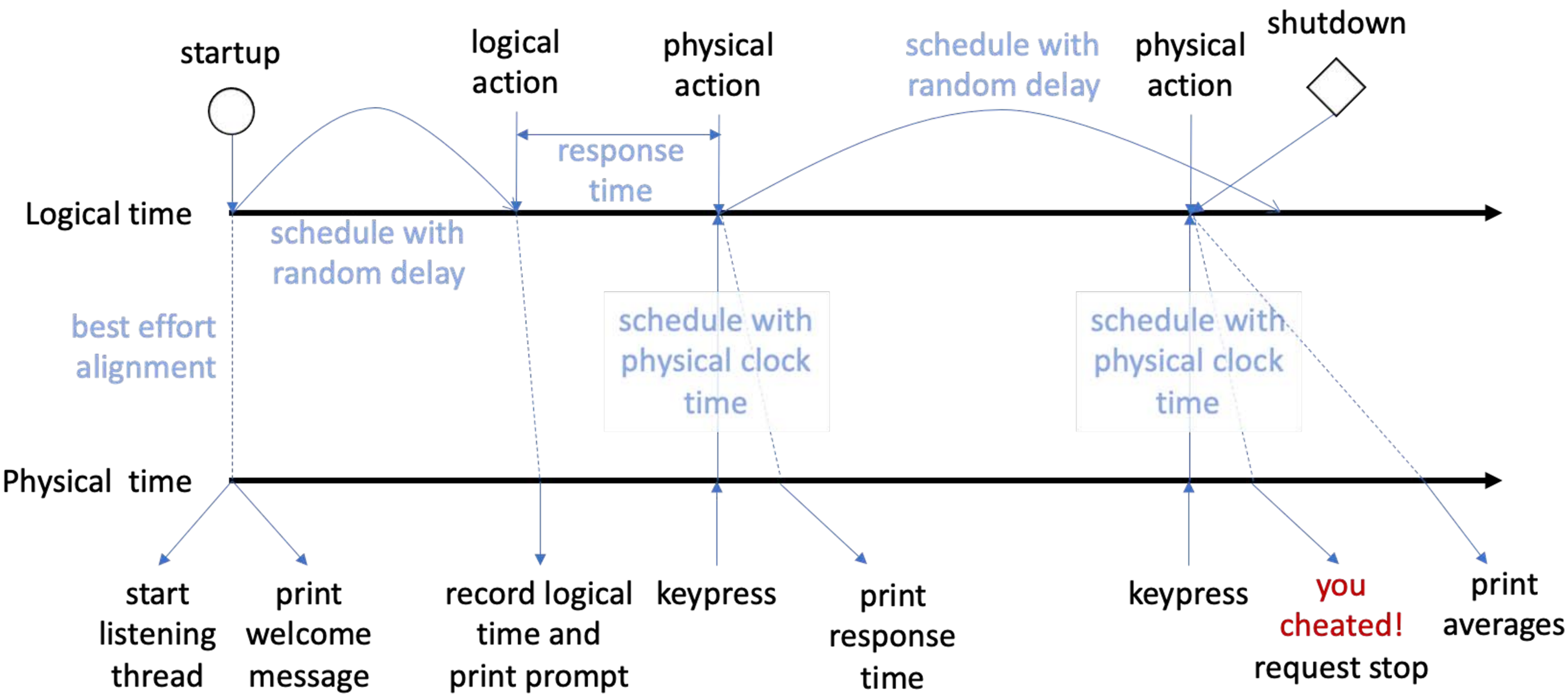
Yes No

Reflex Game

- Reaction Ordering
- Causality Loops
- Physical vs. Logical Time



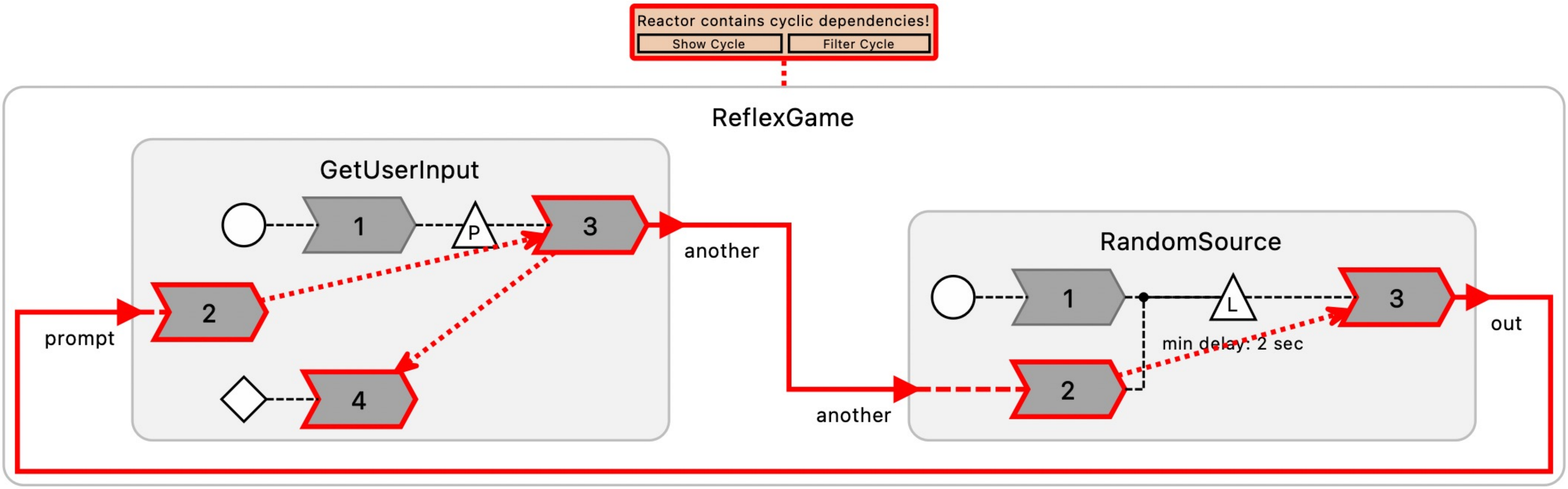
Reflex Game Timing



Causality Loops

If you reverse the order of reactions in the RandomSource, you get a causality loop error.

Why?



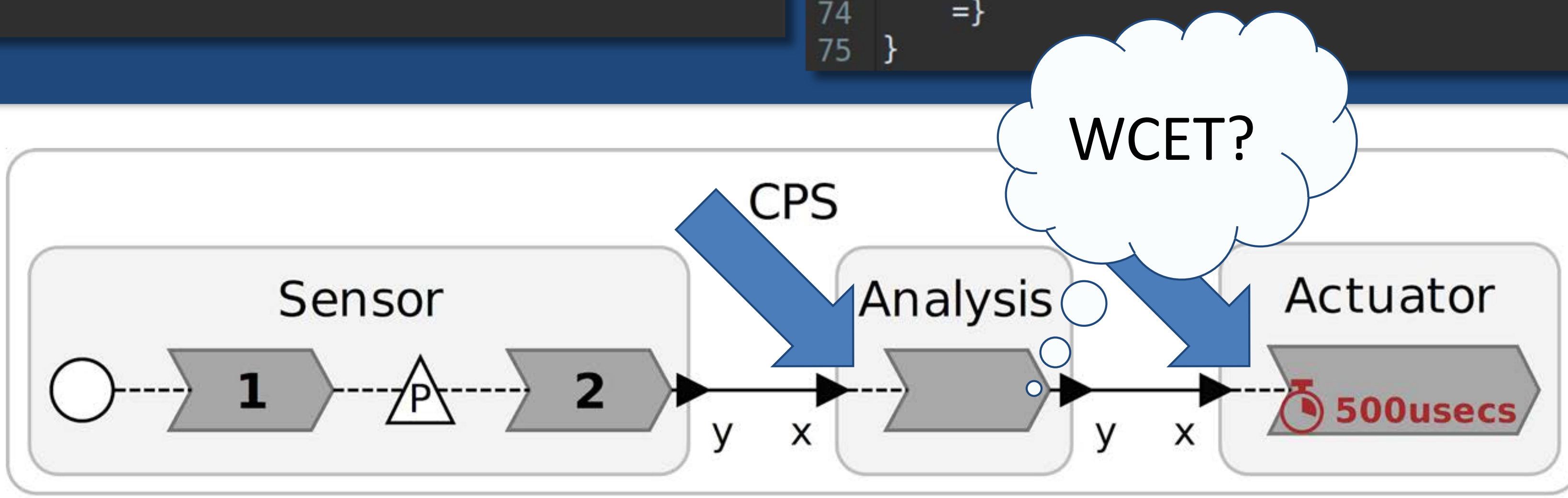
Deadlines

```
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

```
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);
    =}
}
```

T > t+ 500 usec



Preempting Execution

Sieve of Eratosthenes

Anytime computation

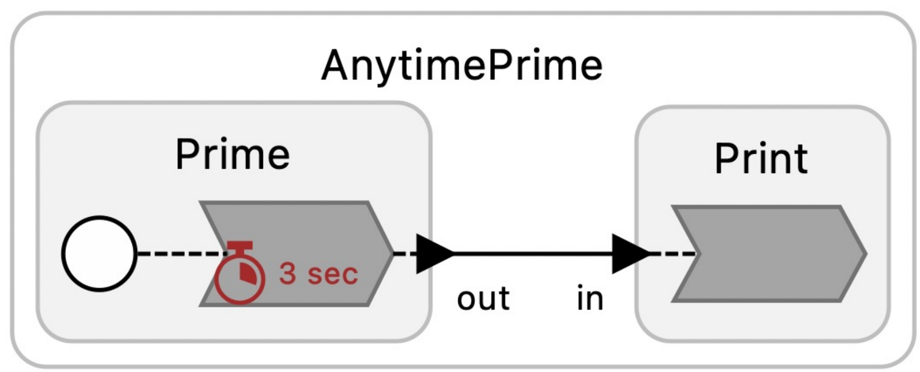
AnytimePrime.lf

```
reactor Prime {
    output out: {=long long=};
    reaction(startup) -> out {=
        int num_primes = 1;
        long long current_num = 2;
        vector_t primes = vector_new(10000);
        vector_push(&primes, (void*)2);

        while (!lf_check_deadline(self, true)) {
            current_num++;
            int i = 0;
            for (i = 0; i < num_primes; i++) {
                if (current_num % (long long)primes.start[i] == 0) {
                    break;
                }
            }
            if (i == num_primes) {
                // Add the prime to vector.
                vector_push(&primes, (void*)current_num);
                num_primes++;
            }
        }

        // Output the largest prime found.
        lf_set(out, (long long)primes.start[num_primes - 1]);
    =} deadline (3 sec) {=
        lf_print("Deadline handler called!");
    =}
}
```

Diagram | Console | Terminal



Possible Deadline Variants

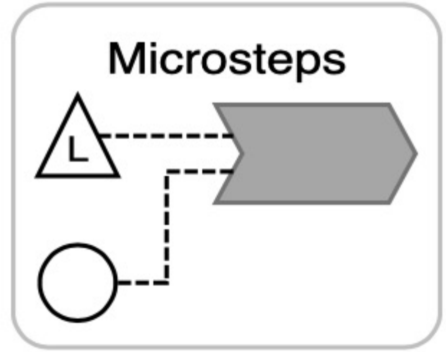
Implemented:

- Lazy deadline
- Cooperative lazy deadline (lf_check_deadline)

Not implemented:

- Eager deadline (Issue #1006)
- Preemptive deadline (kill) (Issue #403)
- Lag trigger (Issue #1006)

Superdense Time




```
target C;
main reactor {
    state count:int(1);
    logical action a;
    reaction(startup, a) -> a {=
        printf("%d. Logical time is %lld. Microstep is %d.\n",
            self->count, lf_tag().time, lf_tag().microstep
        );
        if (self->count++ < 5) {
            lf_schedule(a, 0);
        }
    =}
}
```

```
1. Logical time is 1649607749415269000. Microstep is 0.
2. Logical time is 1649607749415269000. Microstep is 1.
3. Logical time is 1649607749415269000. Microstep is 2.
4. Logical time is 1649607749415269000. Microstep is 3.
5. Logical time is 1649607749415269000. Microstep is 4.
```

Logical simultaneity is a key concept in Lingua Franca.

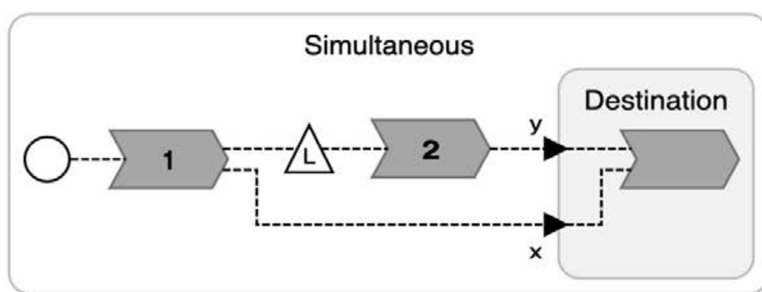
Target language: C

- Resources
- Writing Reactors
  - A First Reactor
  - Inputs and Outputs
  - Parameters and State Variables
  - Time and Timers
  - Composing Reactors
  - Reactions and Methods
  - Causality Loops
  - Extending Reactors
  - Actions
  - Superdense Time
  - Deadlines
  - Multiports and Banks
  - Preambles and Methods
  - Distributed Execution
  - Termination
- Tools
- Reference

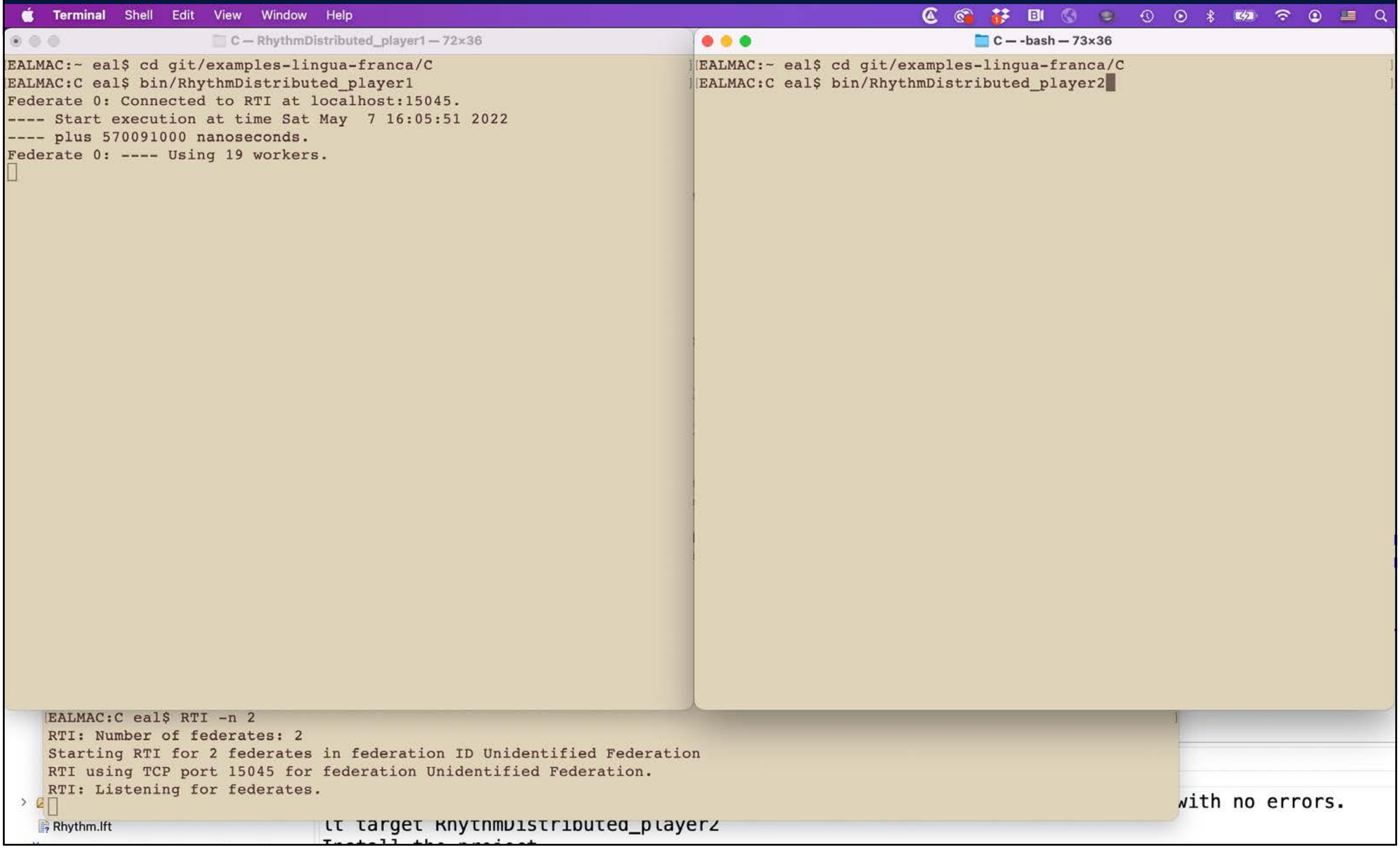
Logical Simultaneity

Two events are **logically simultaneous** only if *both* the logical time and the microstep are equal. The following example illustrates this:

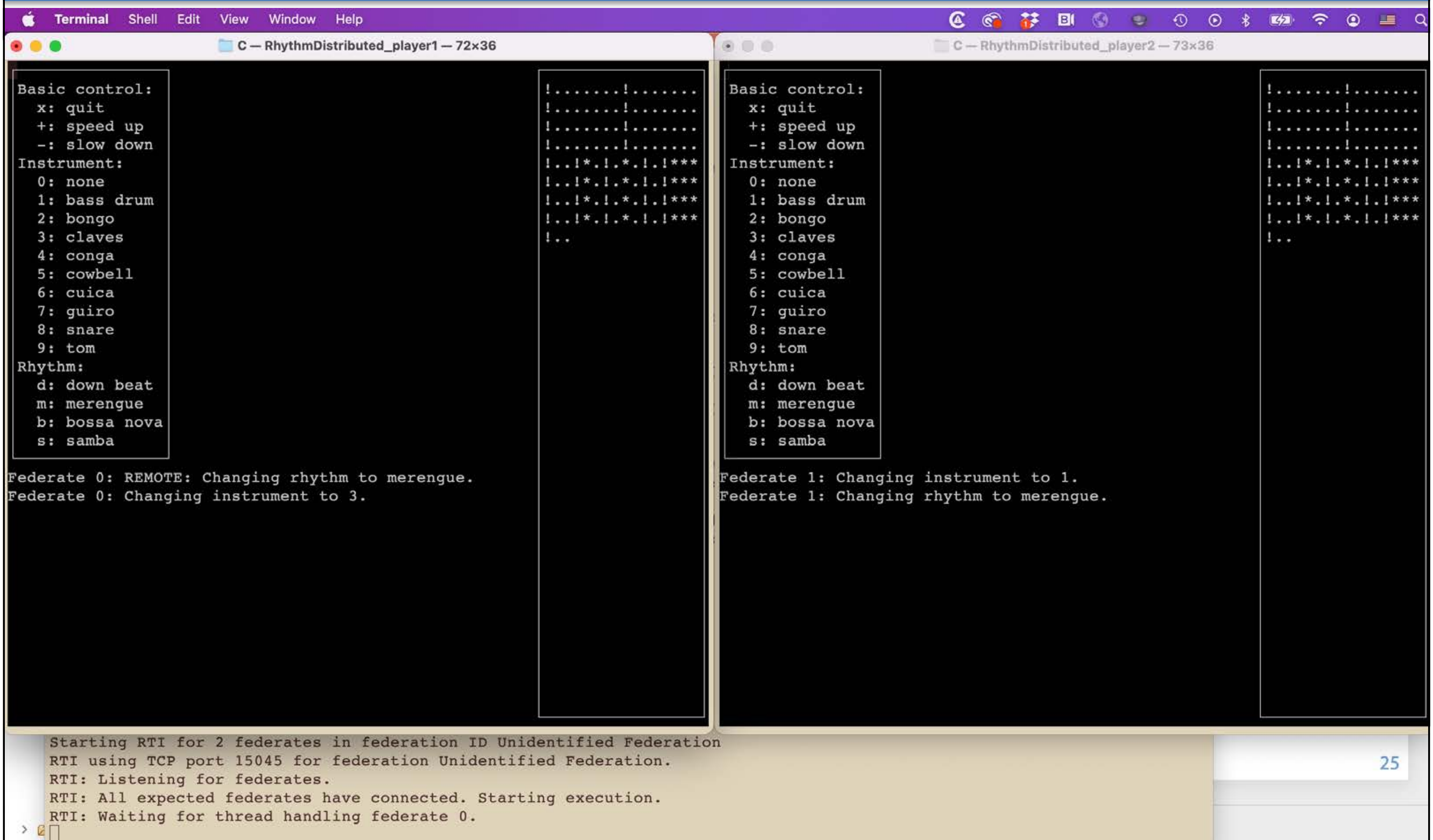
```
target C;
reactor Destination {
    input x:int;
    input y:int;
    reaction(x, y) {=
        printf("Time since start: %lld, microstep: %d\n",
            lf_time_logical_elapsed(), lf_tag().microstep
        );
        if (x->is_present) {
            printf("  x is present.\n");
        }
        if (y->is_present) {
            printf("  y is present.\n");
        }
    =}
}
main reactor {
    logical action repeat;
    d = new Destination();
    reaction(startup) -> d.x, repeat {=
        lf_set(d.x, 1);
        lf_schedule(repeat, 0);
    =}
    reaction(repeat) -> d.y {=
        lf_set(d.y, 1);
    =}
}
```



Teaser: Distributed Rhythm



Teaser: Distributed Rhythm



Conclusion

Concepts:

- Logical time vs. physical time(s)
- Multiple timelines
- Superdense time