TECoSA Seminar, June 3, 2021

iCyPhy

Deterministic Reactive Software for Embedded, Edge, and Cloud Systems

Marten Lohstroh
Advisor: Prof. Edward A. Lee

Computational vs. Reactive Systems

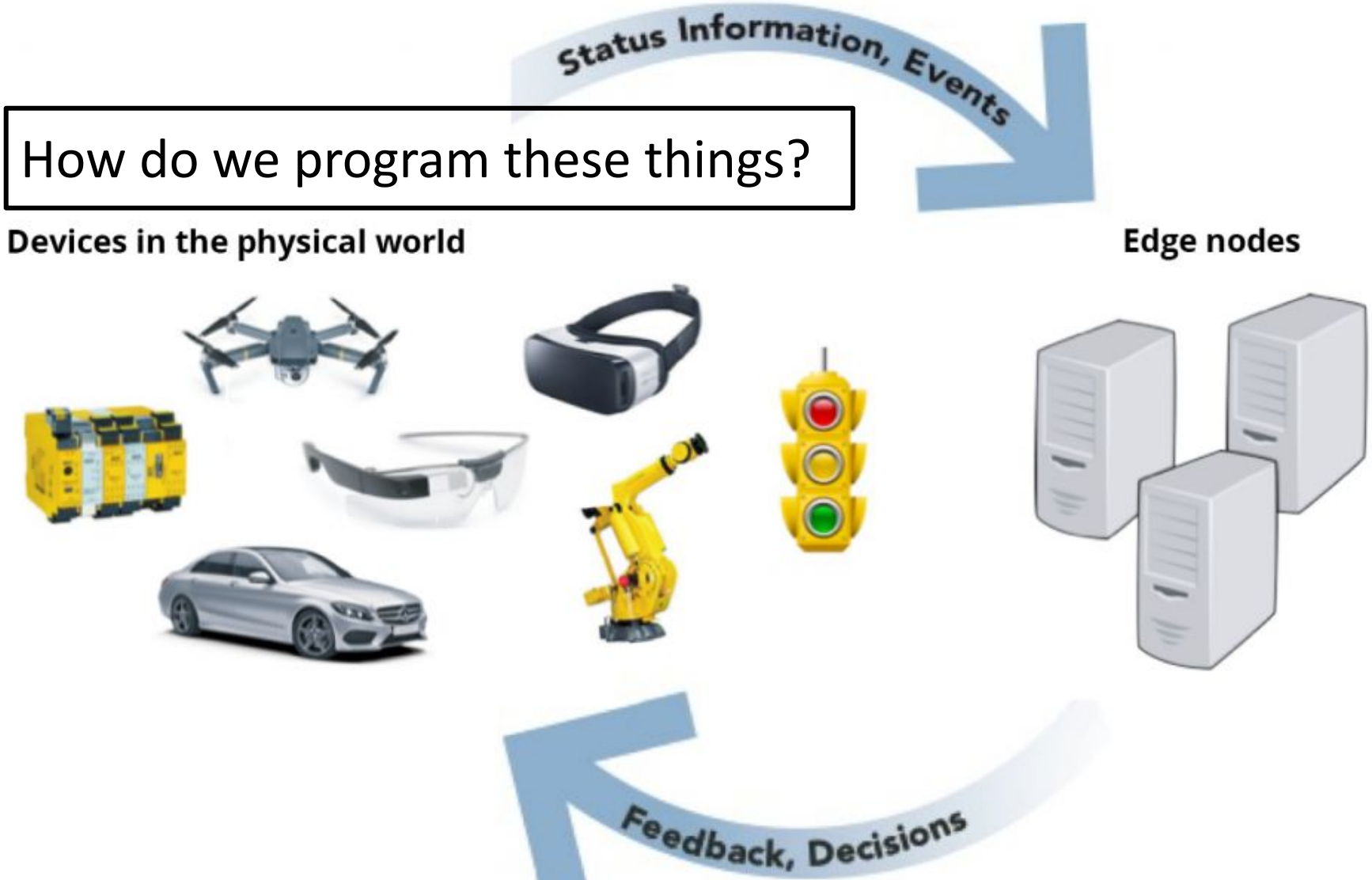
	Computational	Reactive
Behavior	**transformational**	**interactive**
Termination	**goal**	**exception**
Correctness	**input/output**	**trace**
Composition	**sequential/interleaving**	**concurrent**
Time	**performance**	**correctness**

Manna, Z., & Pnueli, A. (1992). Modeling Real Concurrency. In *The Temporal Logic of Reactive and Concurrent Systems* (pp. 103-175). Springer, New York, NY.

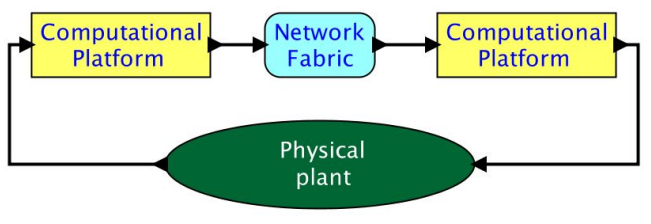
Background image: Drawings of Harrison's H4 chronometer of 1761, published in *The principles of Mr Harrison's time-keeper*, 1767.

Cyber-Physical Systems (CPSs)

How do we program these things?



How do we Ensure Safety in CPS?



Predictability

enables

The major challenge: **Integrating complex subsystems** with adequate **reliability**, **repeatability**, and **testability**.

Safety

Security

Actors

Carl Hewitt

Gul Agha

Unlike previous models of computation, the actor model was inspired by *physics*.

The actor model adopts the philosophy that *everything is an actor*. This is similar to the *everything is an object* philosophy used by some object-oriented programming languages.

An actor is a computational entity that, in response to a message it receives, can concurrently:

- send a finite number of messages to other actors;
- create a finite number of new actors;
- designate the behavior to be used for the next message it receives.

There is no assumed sequence to the above actions and they could be carried out in parallel. (Source: Wikipedia)

Models of Computation (MoCs)

Useful semantics imply constraints on designers — Edward A. Lee

Usable design practice implies: “freedom from choice” — Alberto Sangiovanni-Vincentelli

Photo credit: Rusi Mchedlishvili

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

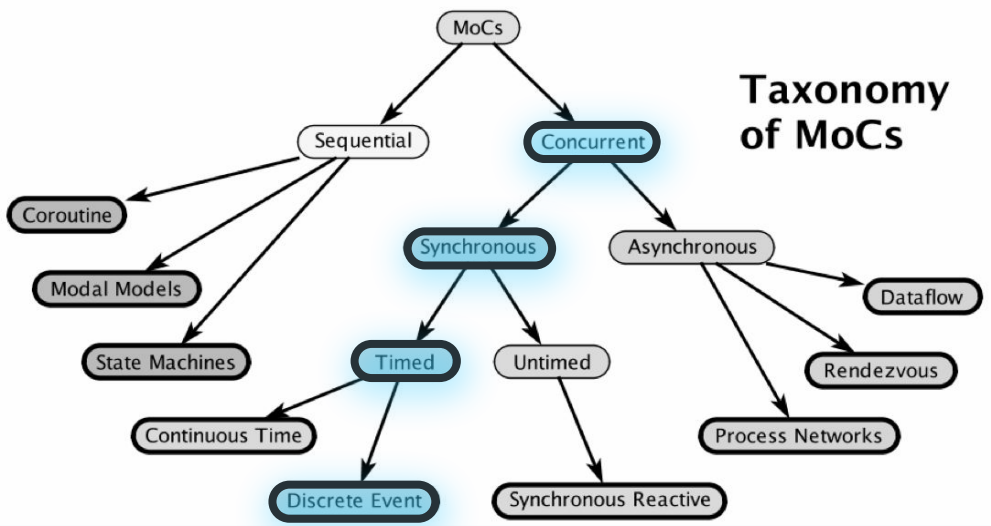
Concurrency, Distribution are Necessary

- Performance, scalability, flexibility, complexity
  - *The "Cyber" part of CPS is getting more complex*
- 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.

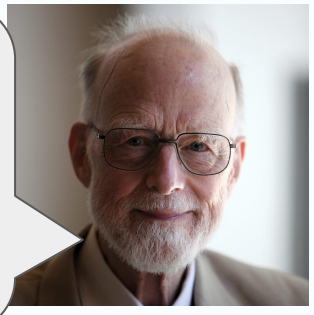
Menard, Christian, et al. "Achieving determinism in adaptive AUTOSAR." *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020.

Leverage Principles of DE

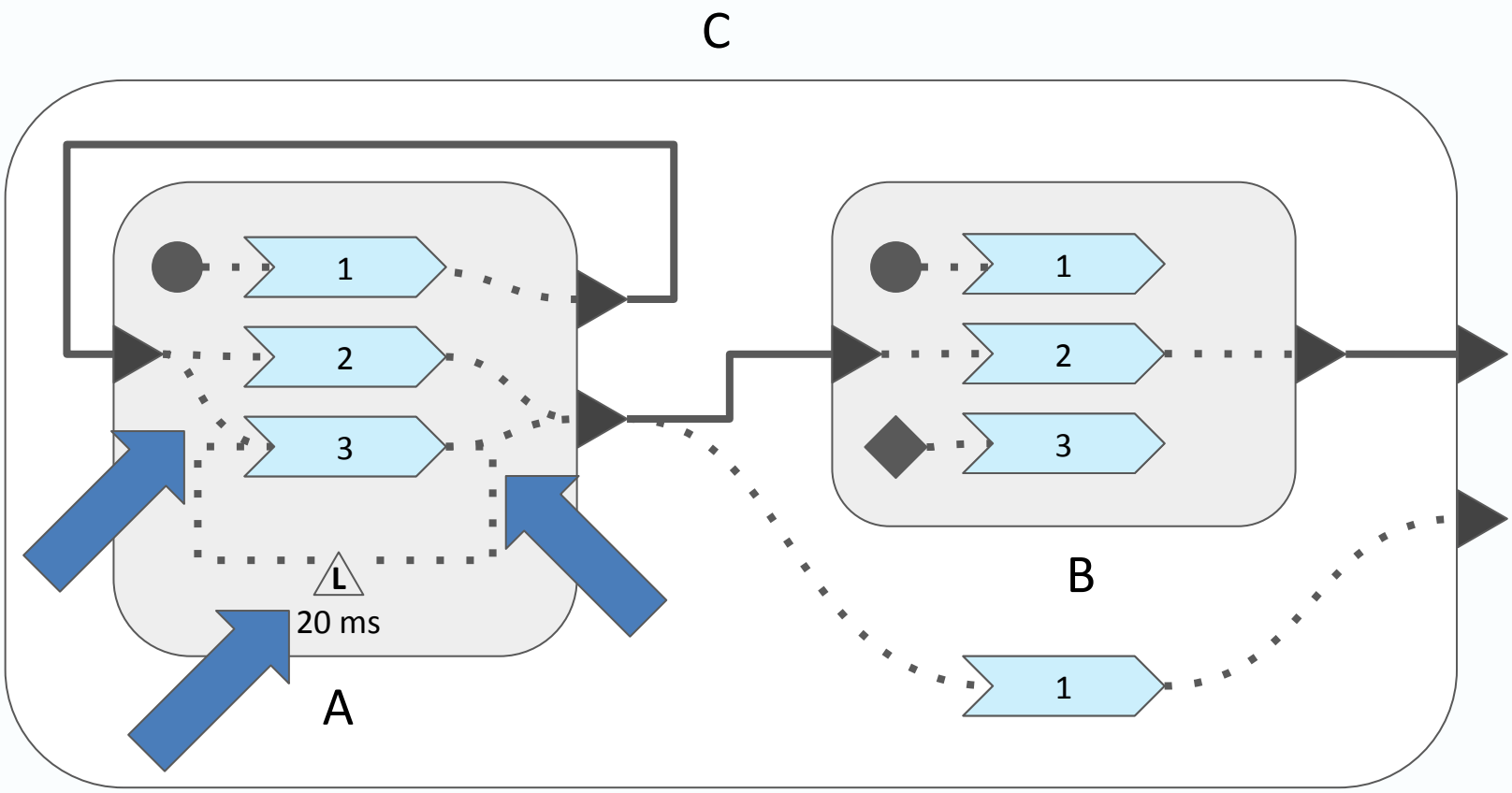
- DE is formally based on SR, but Ptolemy II leverages causality information to avoid expensive fixpoint computation
- Reactors: be smart about the grain at which dependencies are declared and ensure they are conservative but not *too* conservative



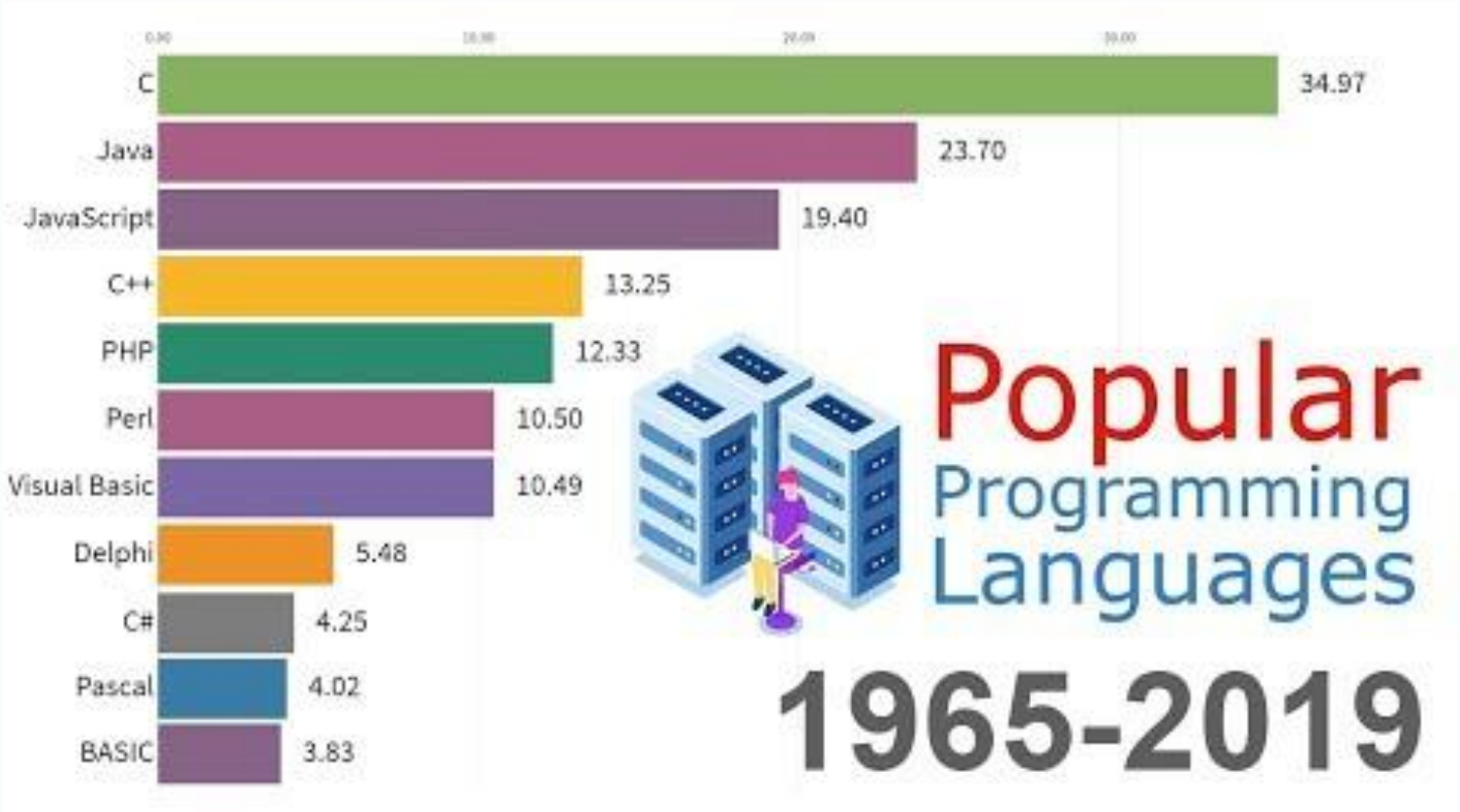
On Language Design: [You] should "have excellent judgment in choosing the best features", but should not "include untried ideas of [your] own. [Your] task is consolidation, not innovation" – C.A.R Hoare



Reactors: (Ptolemy DE) Actors *Revisited*



The PL Popularity Contest



Source: Data is Beautiful (Youtube Channel).
https://www.youtube.com/watch?v=Og847HVwRSI

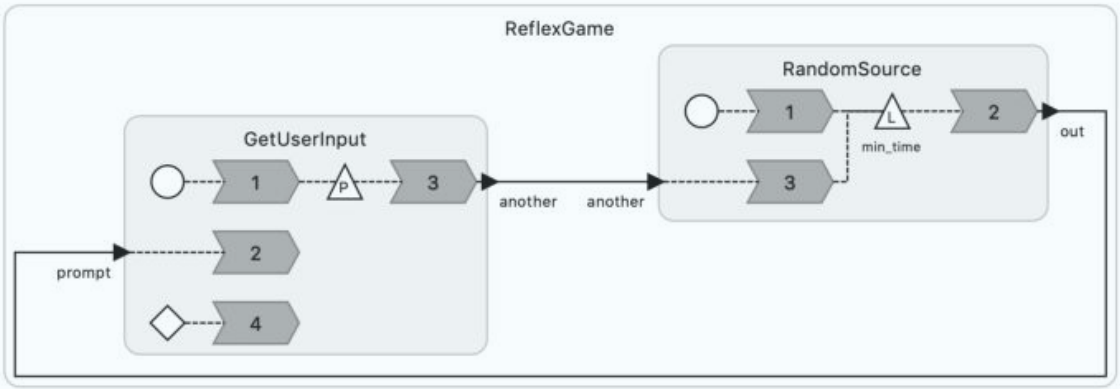
Source: Data is Beautiful (Youtube Channel).
https://www.youtube.com/watch?v=Og847HVwRSI

Lingua Franca: It's About Time

ReflexGame.lf

```
106
107     reaction(shutdown) {=
108         if (self->count > 0) {
109             printf("\n**** Average response time: %d.\n", self->
110         } else {
111             printf("\n**** No attempts.\n");
112         }
113     =}
114 }
115 main reactor ReflexGame {
116     p = new RandomSource();
117     g = new GetUserInput();
118     p.out -> g.prompt;
119     g.another -> p.another;
120 }
121
122
```

Tasks Console Error Log Diagram

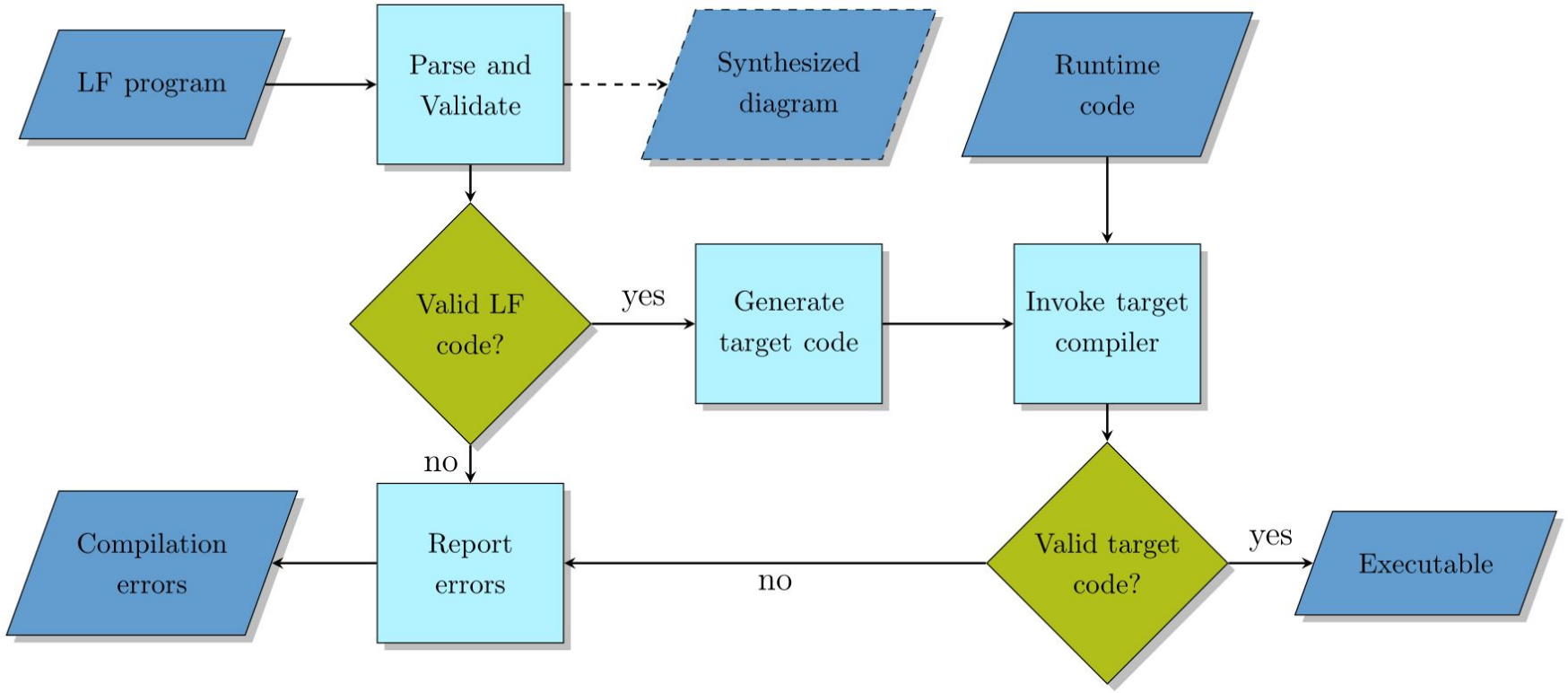


Verbatim target code

Reactor-oriented composition layer

Interactive Visualization

Compiler Toolchain



Logical Time and Physical Time

Logical Time

- Steps or 'ticks'
- Discrete
- Absolute Simultaneity

- External events
- Deadlines
- Federation
- Fault handling

Physical Time

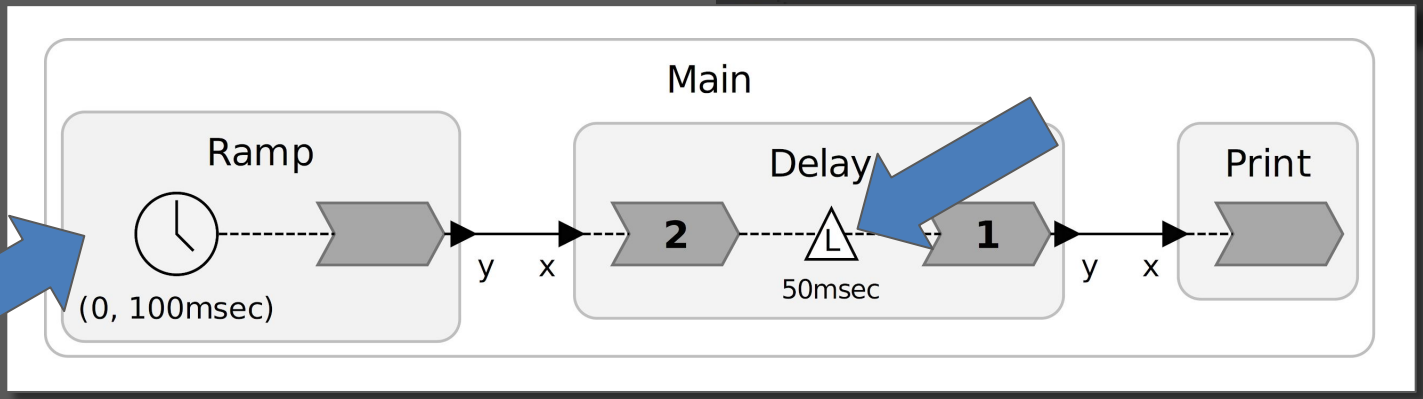
- Measurements
- Continuous
- Relativistic Simultaneity

The Persistence of Memory — Salvador Dalí

Logical Actions

```
1  target C {timeout: 1 sec};
2
3  main reactor Main {
4      ramp = new Ramp();
5      delay = new Delay();
6      print = new Print();
7      ramp.y -> delay.x;
8      delay.y -> print.x;
9  }
10
11 reactor Ramp {
12     timer t(0, 100 msec);
13     output y:int;
14     state count:int(0);
15     reaction(t) -> y {=
16         SET(y, self->count);
17         self->count++;
18     =}
19 }
20
```

```
21 reactor Delay {
22     logical action a(50 msec):int;
23     input x:int;
24     output y:int;
25     reaction(a) -> y {=
26         SET(y, a->value);
27     =}
28     reaction(x) -> a {=
29         schedule_int(a, 0, x->value);
30     =}
31 }
32
33 reactor Print {
34     input x:int;
35     reaction(x) {=
36         printf("Logical time: %lld, Physical time %lld"
37                 ", Value: %d\n",
38                get_elapsed_logical_time(),
39                get_elapsed_physical_time(), x->value);
40     =}
41 }
```



Logical Actions

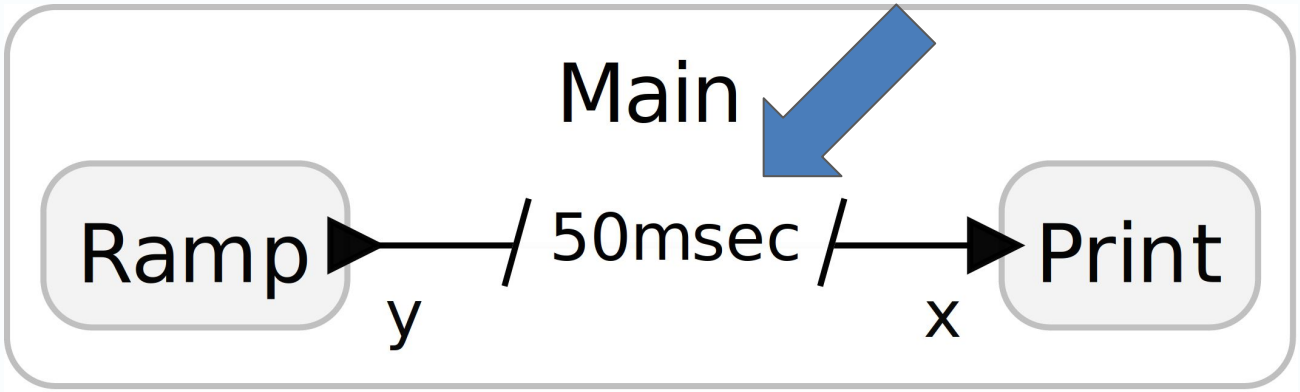
```
[marten@yoga Delay]$ lfc Delay.lf
******** filename: Delay
******** sourceFile: /home/marten/git/lingua-franca/example/Delay/Delay.lf
******** directory: /home/marten/git/lingua-franca/example/Delay
******** mode: STANDALONE
Generating code for: file:/home/marten/git/lingua-franca/example/Delay/Delay.lf
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

```
3  main reactor Main {
4      ramp = new Ramp();
5      delay = new Delay();
6      print = new Print();
7      ramp.y -> delay.x;
8      delay.y -> print.x;
9  }
```

=

```
3  main reactor Main {
4      ramp = new Ramp();
5      print = new Print();
6      ramp.y -> print.x after 50 msec;
7  }
```



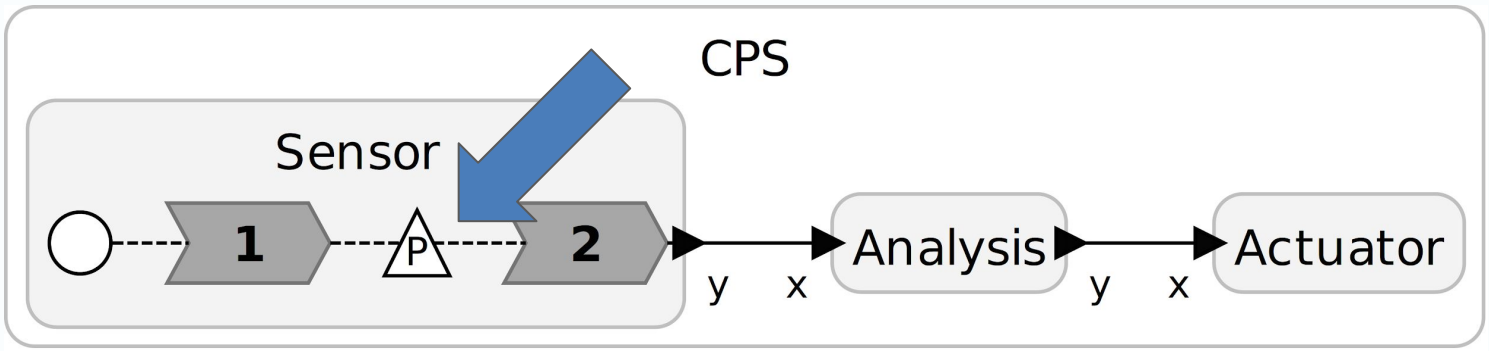
Physical Actions

```
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 inputs, the model defines exactly one behavior.

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

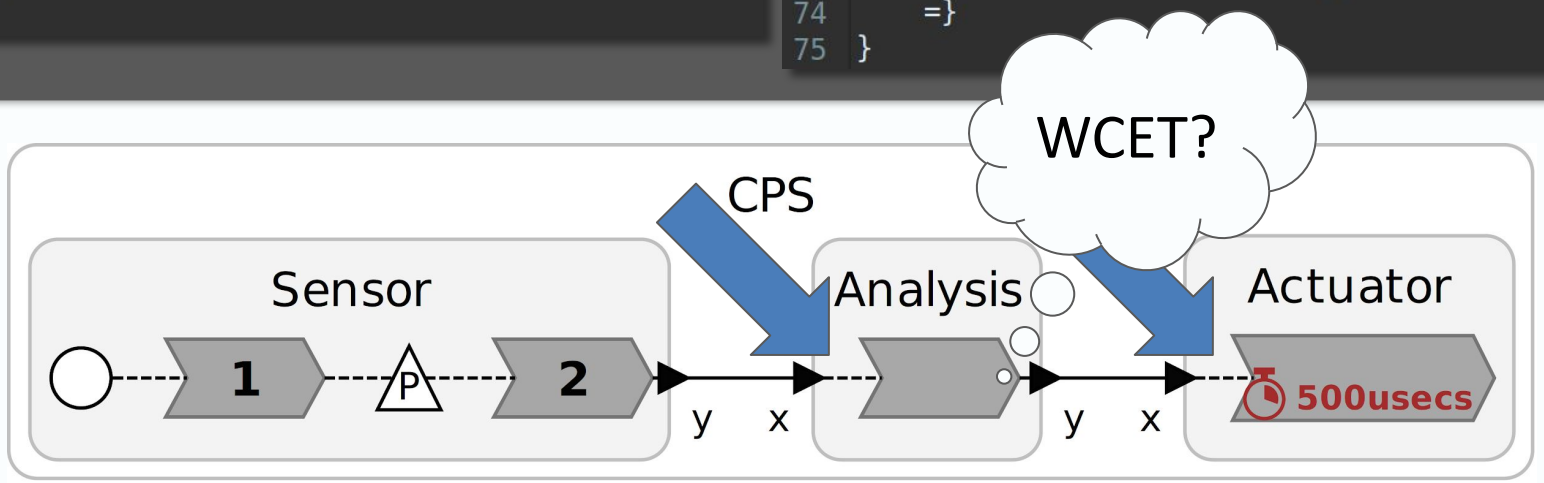
Deadlines

```
44 reactor Analysis {
45     input x:bool;
46     output y:bool;
47     state do_work:bool(false);
48     reaction(x) -> y {=
49         if (self->do_work) {
50             printf("Working for 500 msecs...\n");
51             usleep(500);
52         } else {
53             printf("Skipping work!\n");
54         }
55         self->do_work = !self->do_work;
56         SET(y, true);
57     =}
58 }
```

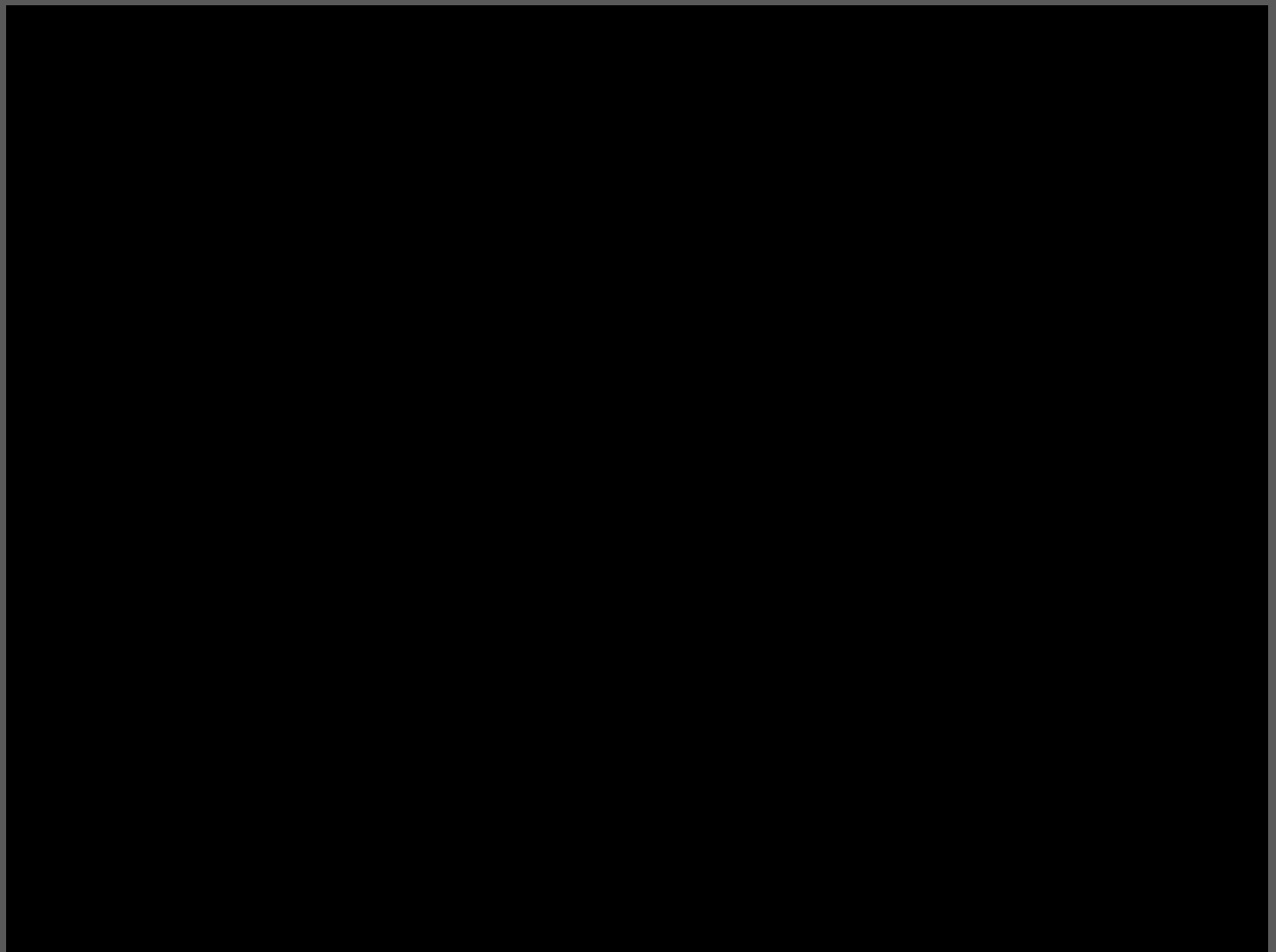
```
60 reactor Actuator {
61     input x:bool;
62     reaction(x) {=
63         instant_t l = get_elapsed_logical_time();
64         instant_t p = get_elapsed_physical_time();
65         printf("Actuating... Logical time: %lld "
66                "Physical time: %lld Lag: %lld\n",
67                l, p, p-l);
68     =} deadline(500 usecs) {=
69         instant_t d = get_elapsed_physical_time()
70                - get_elapsed_logical_time();
71         printf("Deadline missed! Lag: %lld "
72                "(too late by %lld nsecs)\n",
73                d, d-500000);
74     =}
75 }
```

T < t+ 500 usec

T > t+ 500 usec



Deadlines



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*

Scheduling

```
15:  if readyForExec ≠ ∅ then
16:      n ← SELECT(readyForExec); execSet, Q_R ← execSet ∪ {n}, Q_R \ {n}
17:      if Δ(n) = ⊥ ∨ π1(CURRENTTAG()) + Δ(n) < PHYSICALTIME() then
18:          RUNINTHREAD(n)
19:      else
20:          RUNINTHREAD(B_Δ(n))
21:      end if
22:  else
23:      WAITUNTILNUMBEROFIDLETHREADSHASINCREASED()
24:  end if
```

Automatically Exposed Parallelism

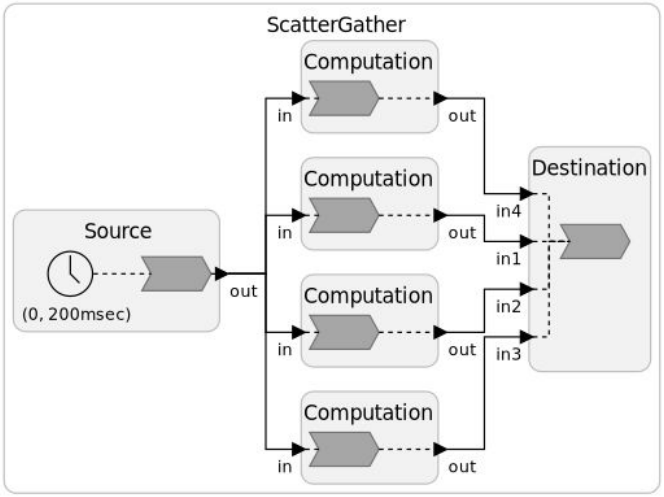


Figure 4.3: Diagram of an LF program realizing a typical scatter/gather pattern.

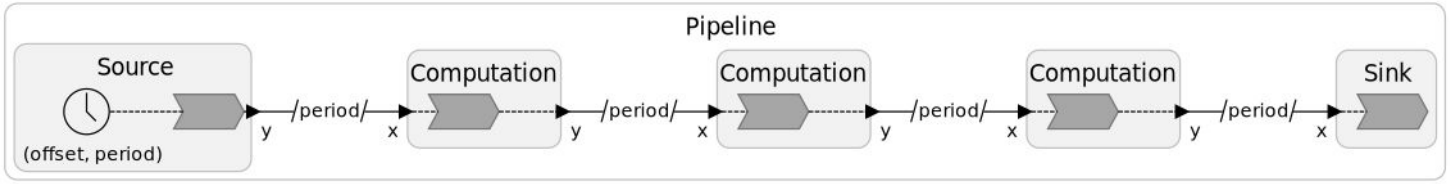
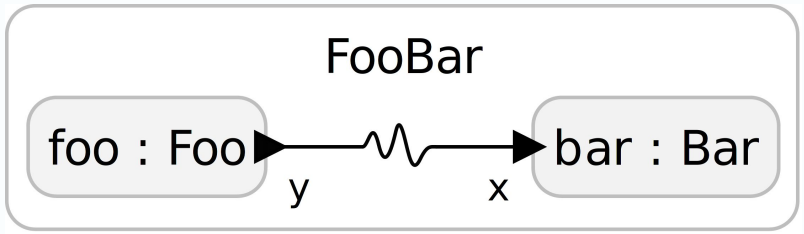


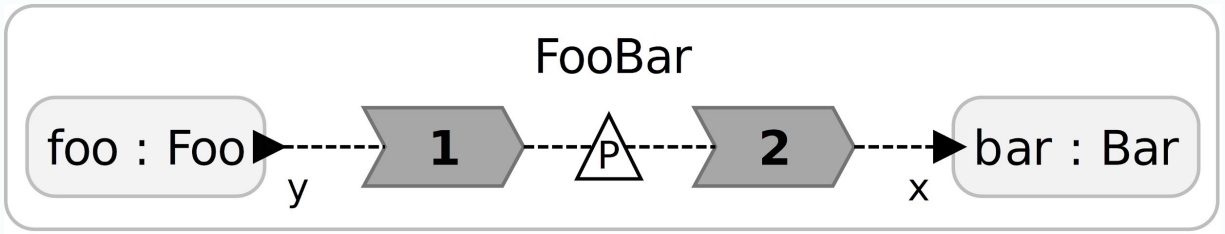
Figure 4.4: Diagram of an LF program that is easy to execute in parallel using pipelining.

Physical Connections




```
reaction (foo.y) -> a {=
        schedule(a, foo.y->value);
=}

reaction (a) -> bar.x {=
    SET(bar.x, a->value);
=}
```



Accessing Dependencies at Runtime

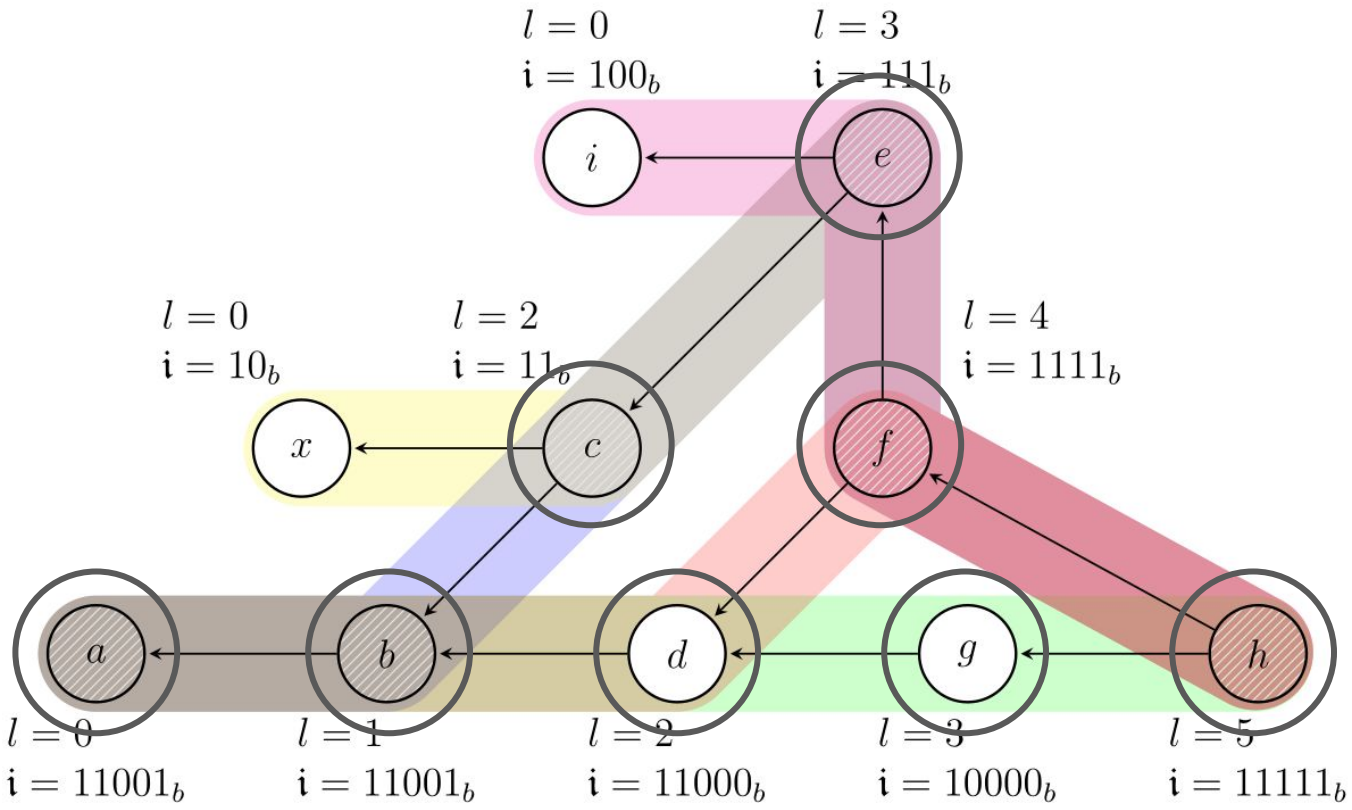
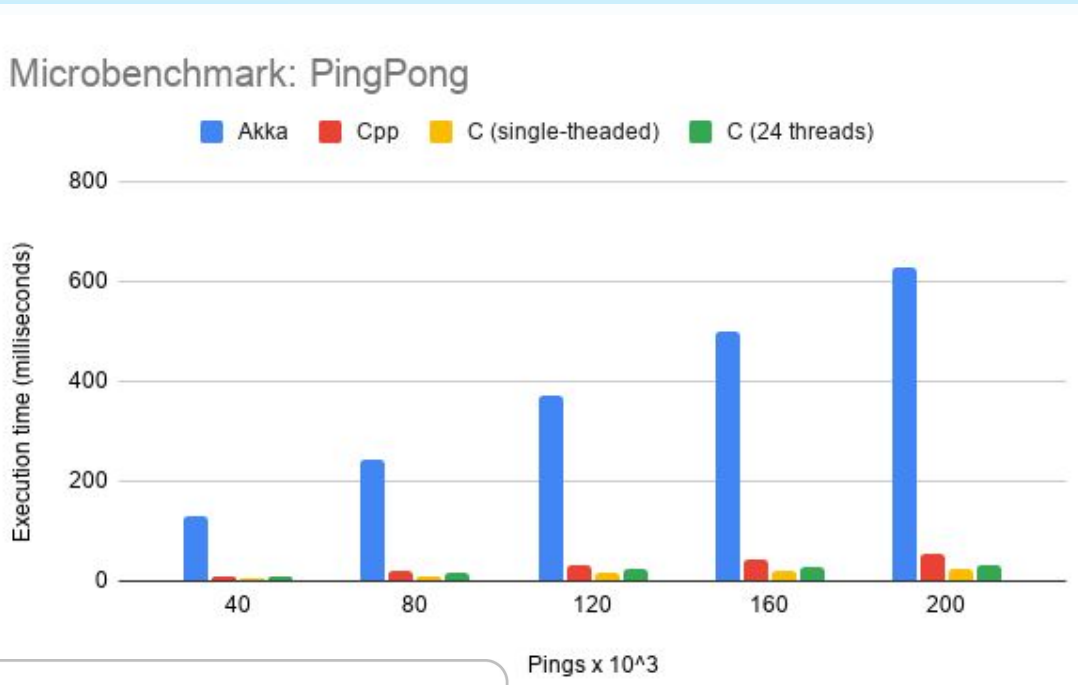
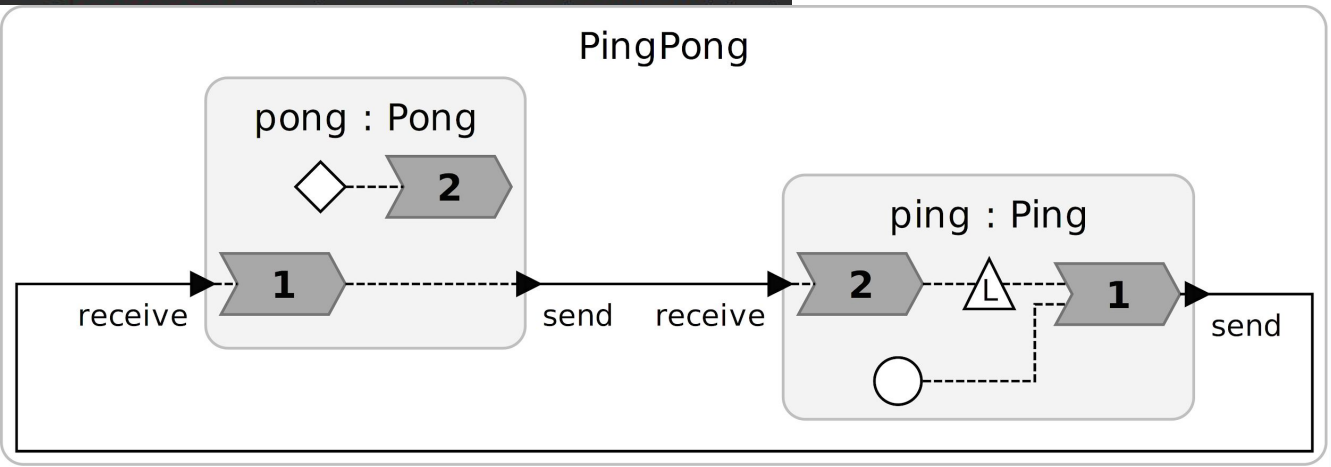


Figure 4.5: Example reaction graph with assigned levels and IDs

Runtime Overhead

```
28 reactor Ping(count:int(1000000)) {
29     input receive:int;
30     output send:int;
31     state pingsLeft:int(count);
32     logical action serve;
33     reaction (startup, serve) -> send
34         SET(send, self->pingsLeft--);
35     =}
36     reaction (receive) -> serve {=
37         if (self->pingsLeft > 0) {
38             schedule(serve, 0);
39         } else {
40             request_stop();
41         }
42     =}
43 }
```

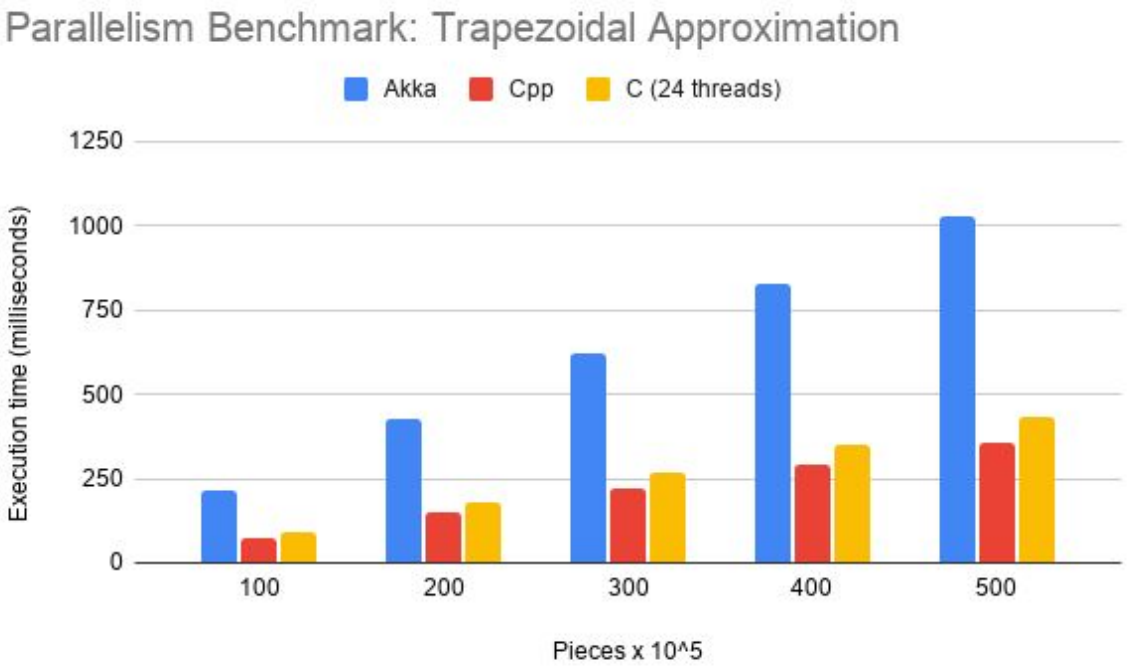


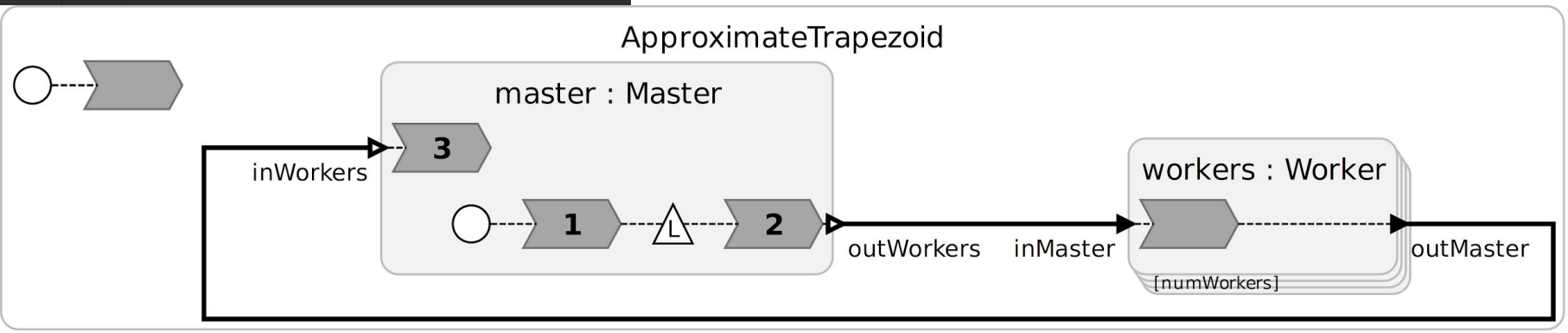


Savina - An Actor Benchmark Suite. Shams Imam, Vivek Sarkar. 4th International Workshop on Programming based on Actors, Agents, and Decentralized Control (AGERE! 2014), October 2014.

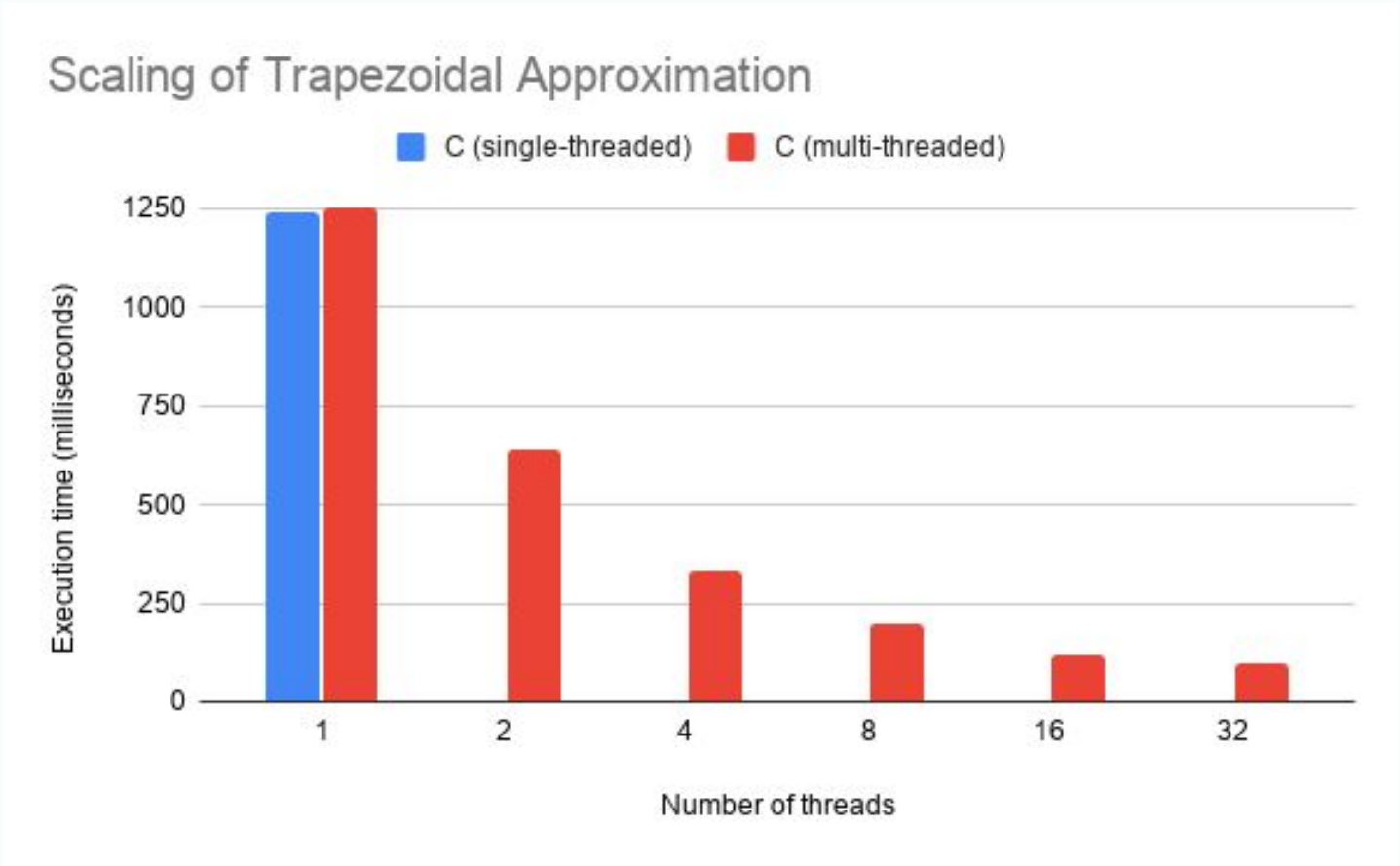
Parallel Execution

```
132
133    reaction(inMaster) -> outMaster {=
134
135        double r = inMaster->value.r;
136        double l = inMaster->value.l;
137        double h = inMaster->value.h;
138        int n = (int)( ((r - l) / h) );
139        double accumArea = 0.0;
140
141        int i = 0;
142        while(i < n) {
143            double lx = (i * h) + l;
144            double rx = lx + h;
145
146            double ly = fx(lx);
147            double ry = fx(rx);
148
149            double area = 0.5 * (ly + ry) * h;
150            accumArea += area;
151
152            i += 1;
153        }
154
155        SET(outMaster, accumArea);
156    =}
```





Scaling on a 6-core Machine (24 hwt)



Mutations

$$A\ reactor\ r\ is\ a\ list\ r = (I, O, \mathcal{A}, S, \mathcal{N}, \mathcal{M}, \mathcal{R}, \mathcal{P}, \{\bullet, \diamond\}),$$

$$\mathcal{M} \subseteq \mathcal{N}\ a\ set\ of\ \boldsymbol{mutations},$$

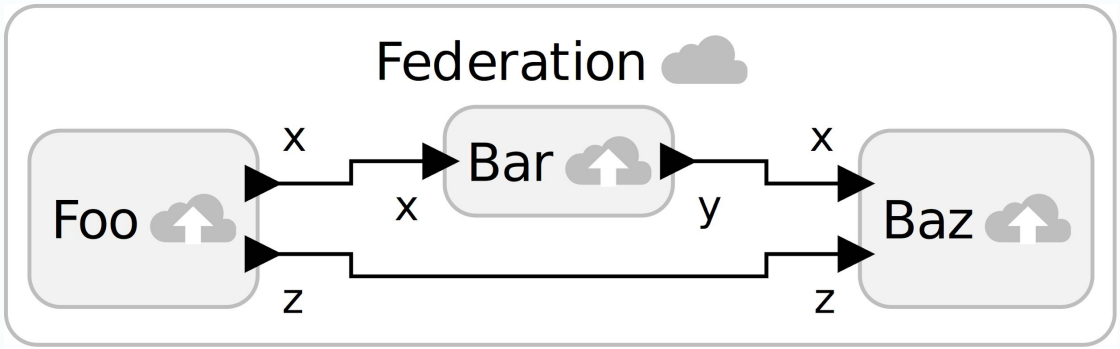
Mutations

Mutations are reactions that have the capability to structurally change a reactor (specifically: $\mathcal{R}$ and $\mathcal{N}$) during the course of its execution. These changes can be carried out using the following API extension that is available to mutations:

- CREATE: Creates a new reactor instance given a reference to a reactor class;
- DELETE: Deletes the reactor identified by a given references from its container;
- CONNECT: Connects the ports of two reactors; and
- DISCONNECT: Disconnects the ports of two reactors.

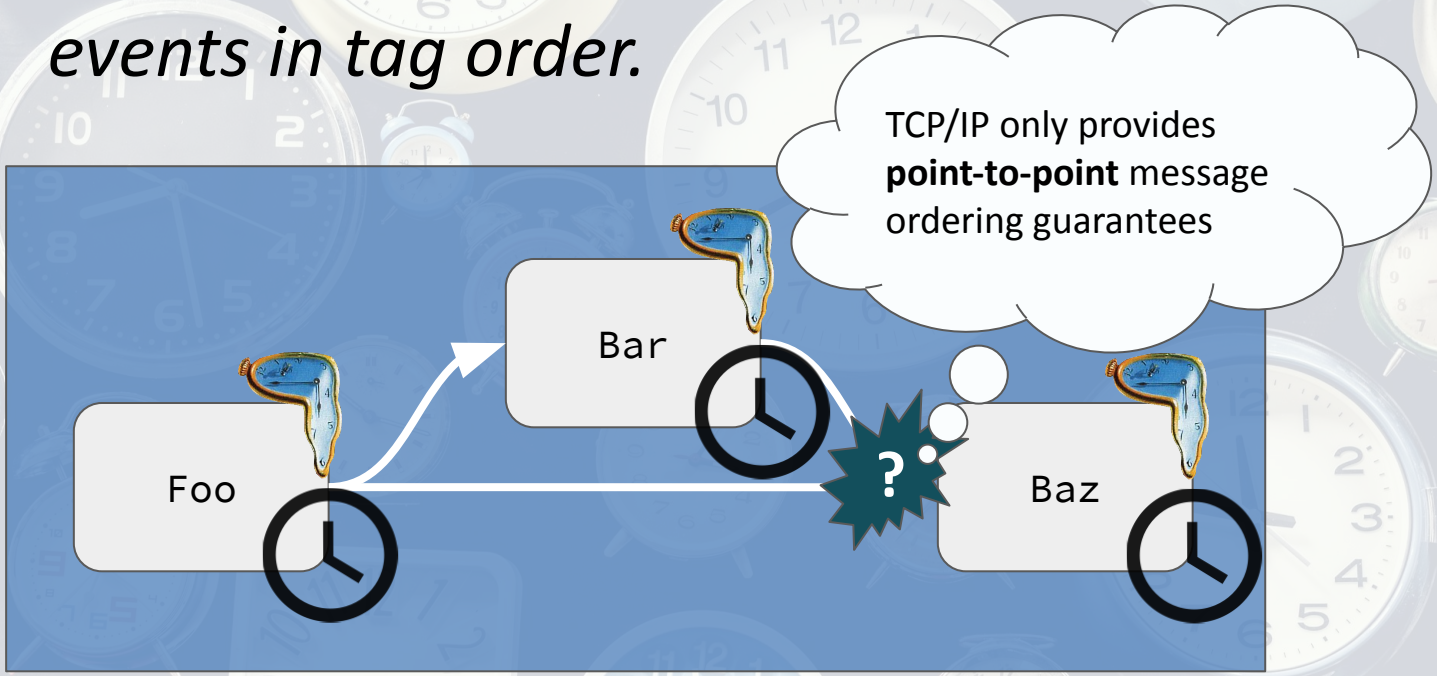
Federated LF Programs

```
1  target C;
2
3  federated reactor Federation at localhost:15044 {
4      foo = new Foo() at foo.host:99999;
5      bar = new Bar() at bar.host:99999;
6      baz = new Baz() at baz.host:99998;
7
8      foo.x -> bar.x;
9      foo.z -> baz.z;
10     bar.y -> baz.x;
11 }
```



Federation: A Multiplicity of Timelines

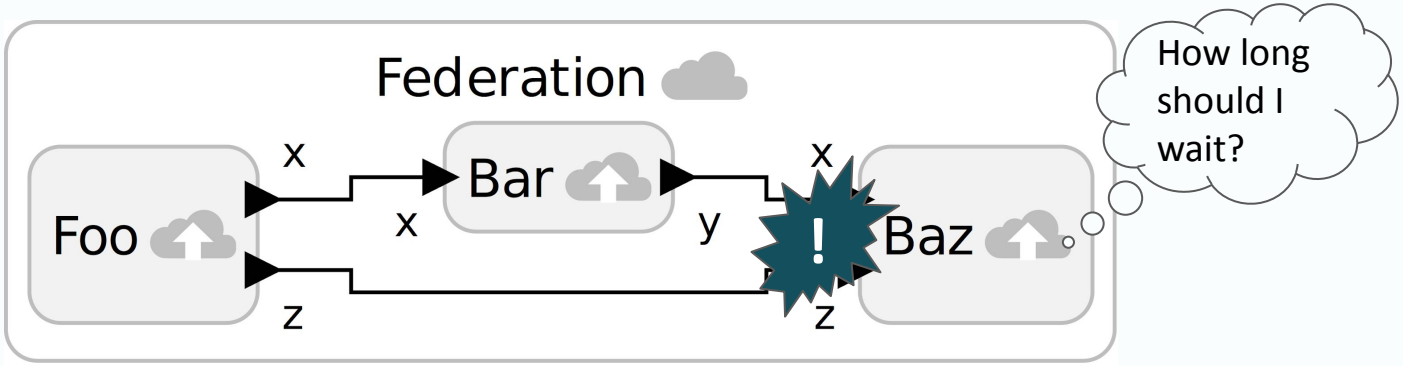
***Goal**: have each federate observe all tagged events in tag order.*

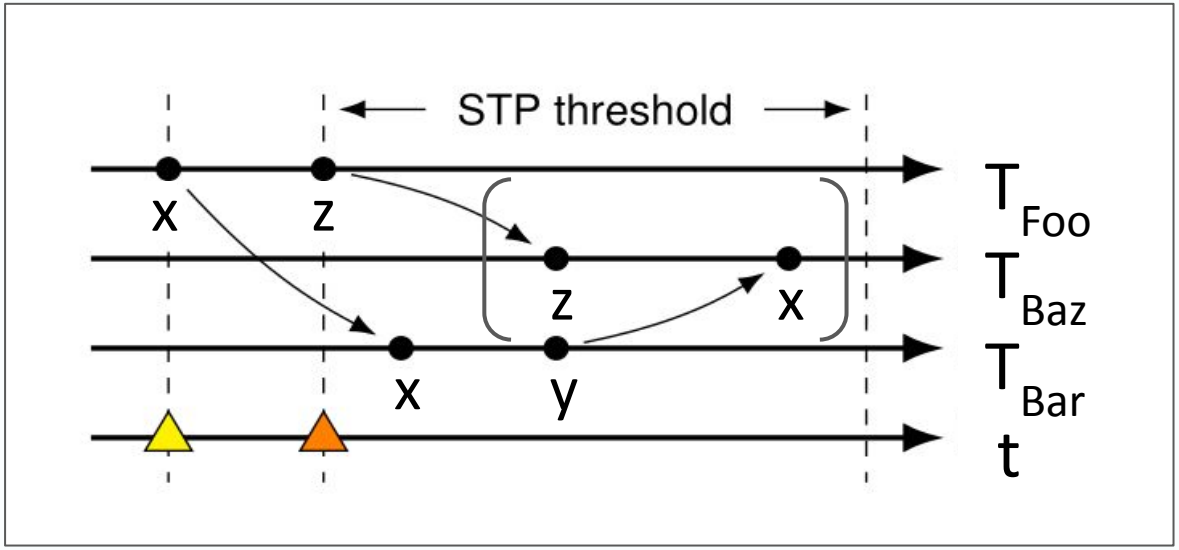


Centralized Coordination

- Central coordinator
  - controls advancement of logical time
  - relays messages between federates
  - forms a performance bottleneck
  - ...and also a single point of failure

Distributed Coordination: PTIDES[1]





Safe-to-process threshold assumes bounds on:

- Execution times
- Network latency
- Clock synchronization error

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

Time: Not Only For Real-Time Systems

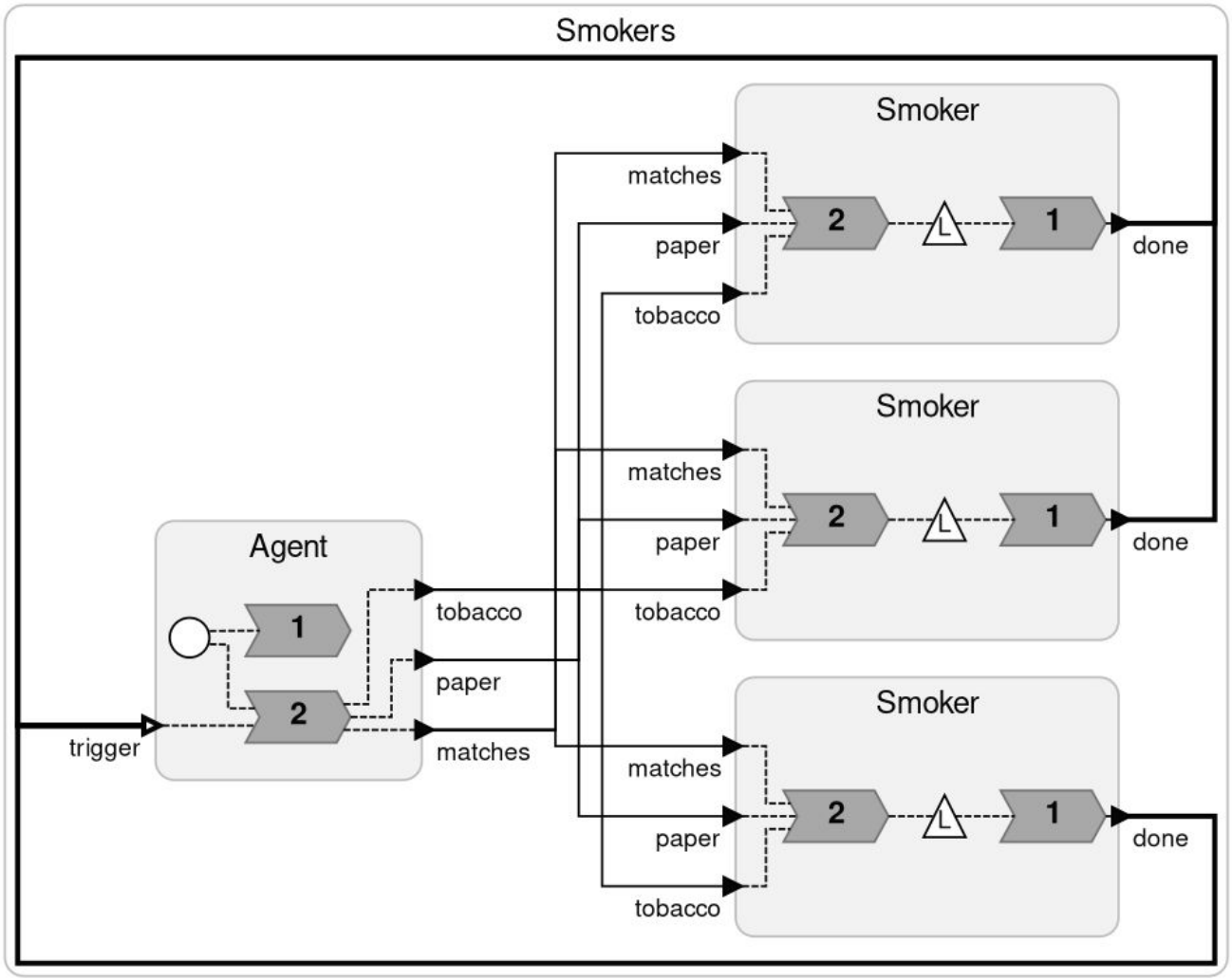
- A semantics of logical time provides a natural framework for reasoning about concurrency
- Makes some difficult problems easy
- Enables quantified evaluation of the tradeoff between *consistency* and *availability*

Example: Cigarette Smokers Problem[1]

[1] Due to Suhas Patil (1971)

- A cigarette requires three ingredients to make and smoke: tobacco, paper, and matches.
- Each smoker has an infinite supply of *one* ingredient
- An agent arbitrarily puts two ingredients on the table, waits until one smoker has smoked
- Each smoker has to acquire two locks before being able to smoke; *How to avoid deadlock?*

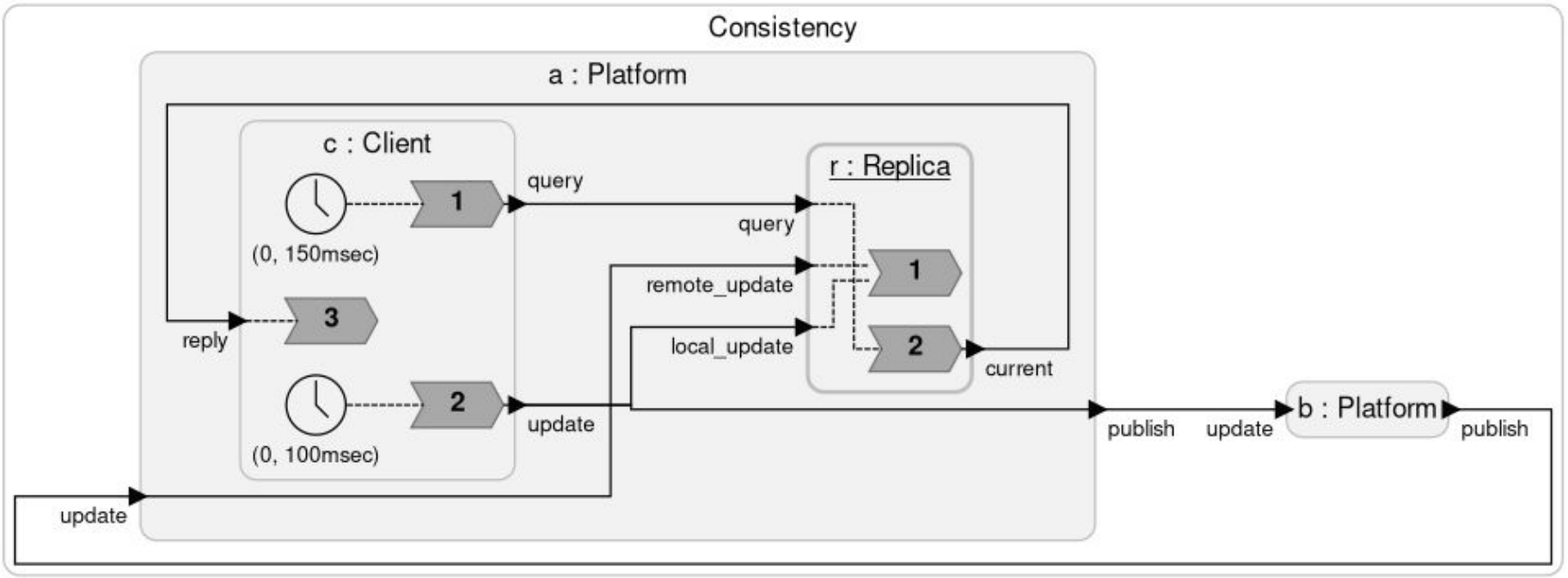
Cigarette Smokers in Lingua Franca



- Smokers execute concurrently
- Logical notion of **simultaneity** drastically simplifies the required coordination logic!

Example: Replicated Database[1]

[1] Inspired by Lamport (1984)



Strong **consistency**. When federated, this comes **at the cost of availability** (i.e., a physical time delay) bounded from below by the network latency between the two platforms!

Future Ongoing/Work

- Expand on quantified CAP Theorem
- More thorough performance evaluation
- Improve runtime performance
- More robust distributed execution
- Integration with ROS/AUTOWARE
- Security
- Implement runtime support for mutations
- Target bare-metal FlexPRET
- Preemptive EDF
- WCET tools in the compiler?

Future Ongoing/Work (Continued)

- Rust target
- Integration with AUTOSAR
- Support for other IDEs through LSP
- LF syntax for modal models
- Establish regular release cycle/nightly builds
- Improve documentation
- Website (Coming soon!)
- ...

Conclusion

- Reactors/Lingua Franca can augment mainstream programming languages with:
  - deterministic concurrency based on synchronous-reactive principles
  - a rich model of time
  - scheduler that can handle periodic tasks as well as sporadic events
  - deadlines
  - high-performance, automatic parallelism
  - federated execution (in progress)
  - deterministic runtime mutations (future work)

Acknowledgements

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

Others who have influenced LF with their ideas (in alphabetical order) are: Abanob Bostouros, Janette Cardoso, Jeronimo Castrillon, Julien Deantoni, Patricia Derler, Clement Fournier, Christopher Gill, Andrés Goens, Reinhard von Hanxleden, Hannes Klein, Zheng Liang, Íñigo Íncer Romeo, Marcus Rossel, Alberto Sangiovanni-Vincentelli, Martin Schoeberl, Sanjit Seshia, Marjan Sirjani, Edward Wang, Felix Wittwer, and Sheng-Jung Yu.

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.

Check Out LF!

LINGUA FRANCA

GitHub

repo.lf-lang.org

Discourse

community.lf-lang.org