iCyPhy

Software Design for Cyber-Physical Systems

Edward A. Lee

Module 6: Parallel Execution

Technical University of Vienna
Vienna, Austria, May 2022

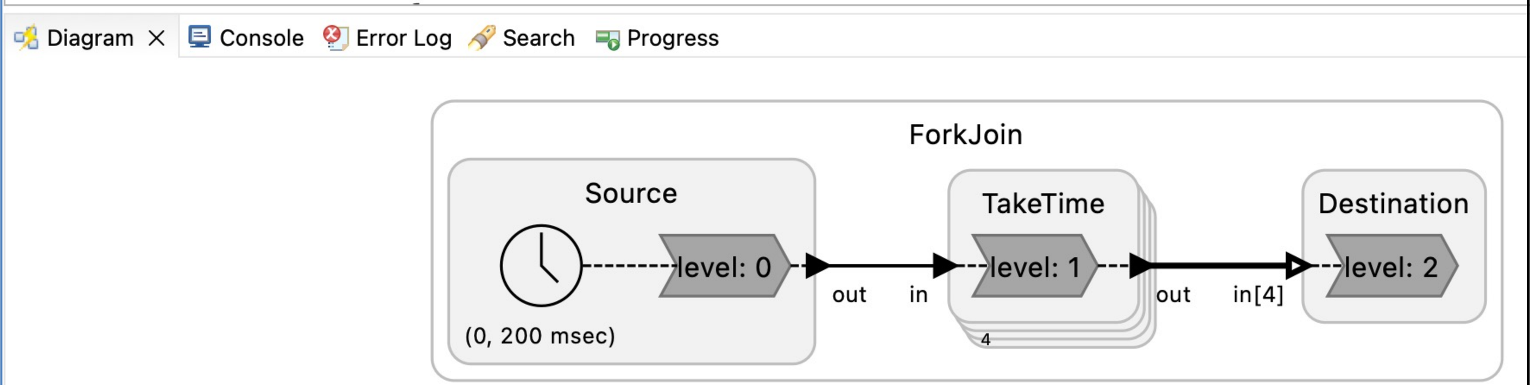
University of California, Berkeley

Parallelism

Multicore execution preserves deterministic semantics.

ForkJoin.lf

```
/**
 * Each instance of TakeTime takes 200 ms wall clock time to
 * transport the input to the output. Four of them are
 * instantiated. Note that without parallel execution, there is
 * no way this program can keep up with real time since in every
 * 200 msec cycle it has 800 msec of work to do. Given 4 workers,
 * however, this program can complete 800 msec of work in about
 * 225 msec.
 */
target C {
    timeout: 2 sec,
    workers: 1, // Change to 4 to see speed up.
};
main reactor(width:int(4)) {
    a = new Source();
    t = new[width] TakeTime();
    (a.out)+ -> t.in;
    b = new Destination(width = width);
    t.out -> b.in;
}
```

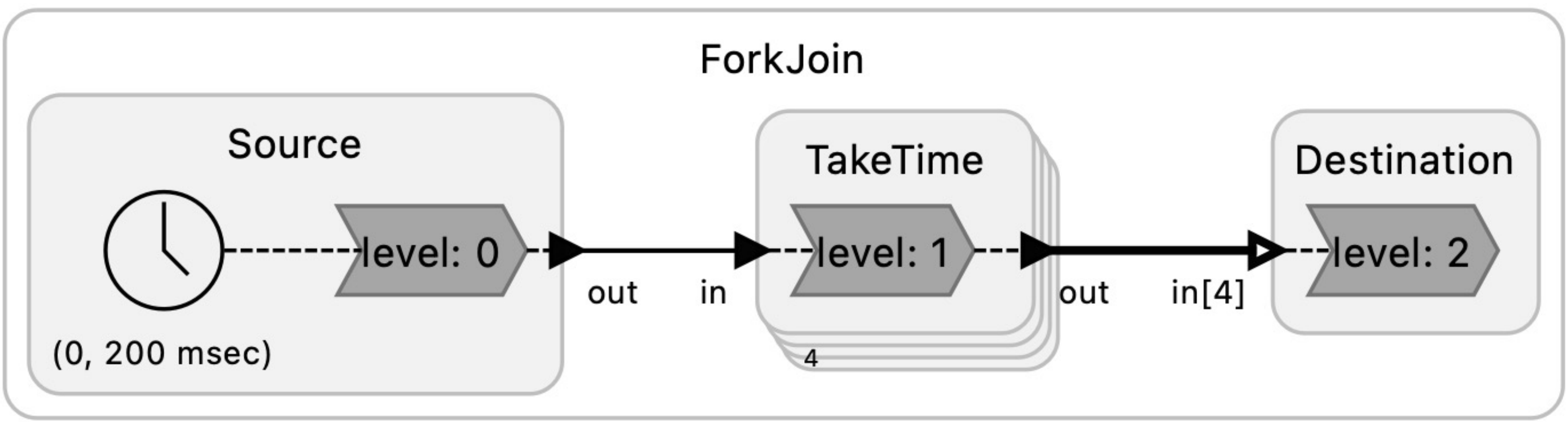


Event and Reaction Queues

Event queue, sorted by tag

Reaction queue, sorted by deadline and level.

Worker 1	Worker 2	Worker 3	Worker 4



Pipeline

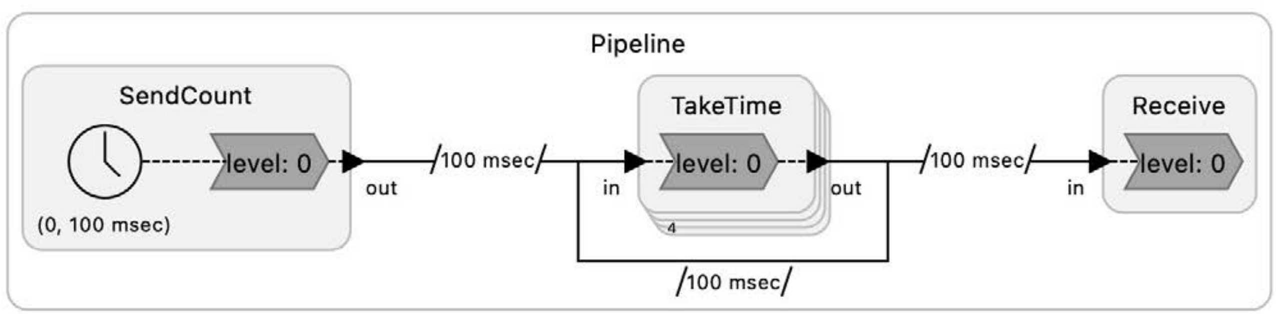
To get parallelism, the pipeline pattern requires careful attention to tags.

Pipeline.lf

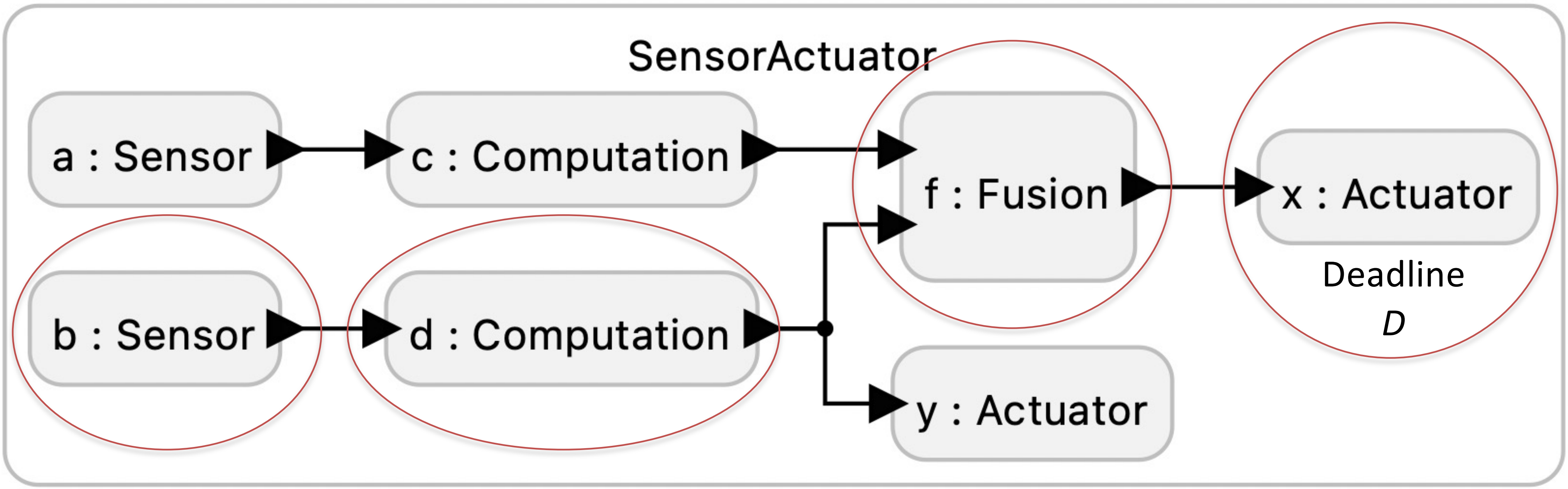
```
/**
 * Basic pipeline pattern where a periodic source feeds
 * a chain of reactors that can all execute in parallel
 * at each logical time step.
 *
 * The workers argument specifies the number of worker
 * workers, which enables the reactors in the chain to
 * execute on multiple cores simultaneously.
 *
 * This uses the TakeTime reactor to perform computation.
 * If you reduce the number of worker workers to 1, the
 * execution time will be approximately four times as long.
 *
 * @author Edward A. Lee
 * @author Marten Lohstroh
 */
target C {
    workers: 4,
}

main reactor {
    r0 = new SendCount(period = 100 msec);
    rp = new[4] TakeTime(approximate_time = 100 msec);
    r5 = new Receive();
    // Comment the "after" clause to eliminate parallelism.
    r0.out, rp.out -> rp.in, r5.in after 100 msec;
}
```

Diagram | Console | Error Log | Search | Progress | Terminal



Motivating Example



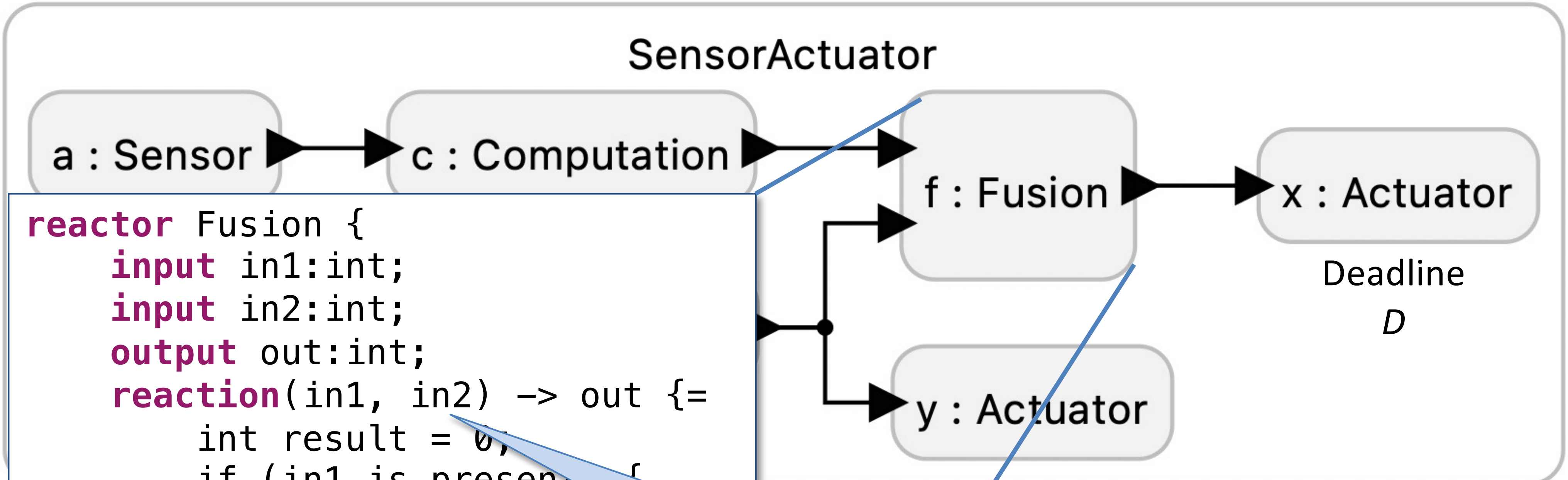
Sporadic events are assigned a time stamp based on the local physical-time clock

Computations have logically zero delay.

Every reactor handles events in time-stamp order. If time-stamps are equal, events are "simultaneous"

Actuators can have a deadline *D*. An input with time stamp *t* is required to be delivered to the actuator before the local clock hits *t* + *D*.

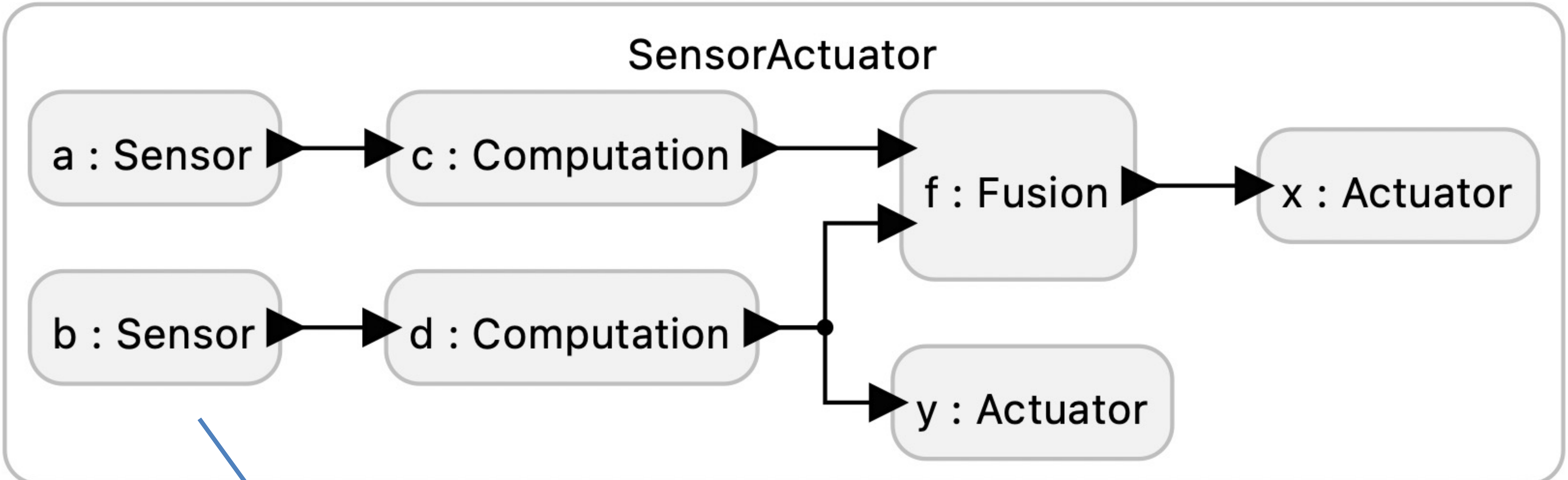
Determinism




```
reactor Fusion {
    input in1:int;
    input in2:int;
    output out:int;
    reaction(in1, in2) -> out {=
        int result = 0;
        if (in1_is_present) {
            result += in1;
        }
        if (in2_is_present) {
            result += in2;
        }
        set(out, result);
    =}
}
```

Whether the two triggers are present simultaneously depends only on their timestamps, not on when they are received nor on where in the network they are sent from.

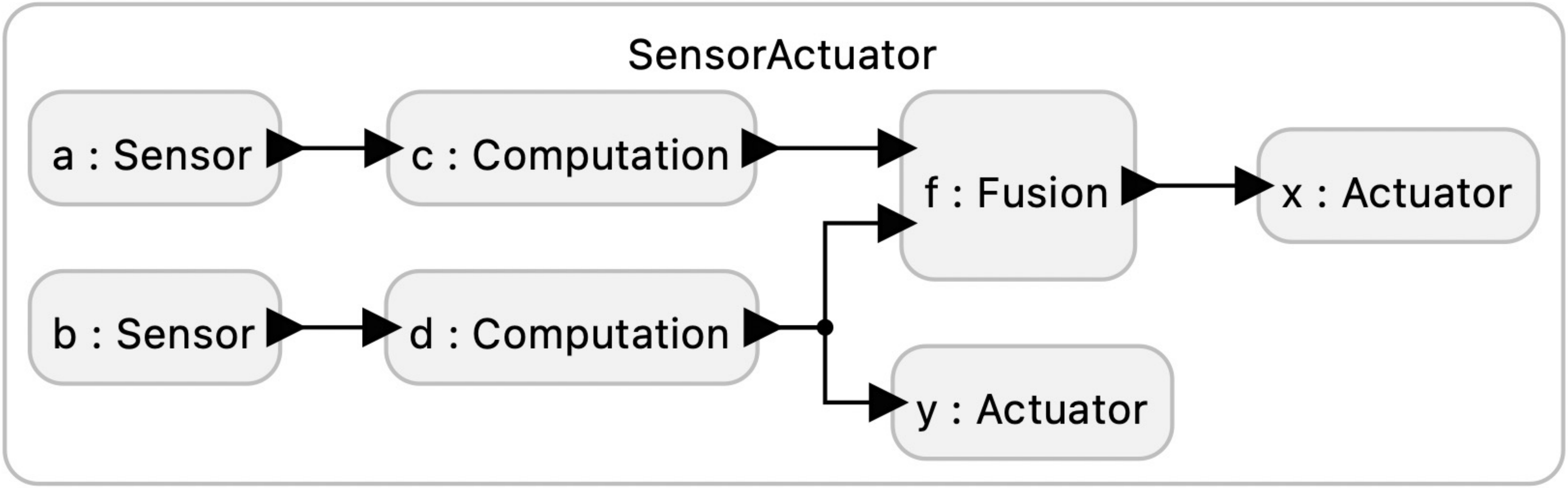
Simple, Sequential Execution



When a sporadic sensor triggers (or an asynchronous event like a network message arrives), assign a time stamp based on the local physical-time clock.

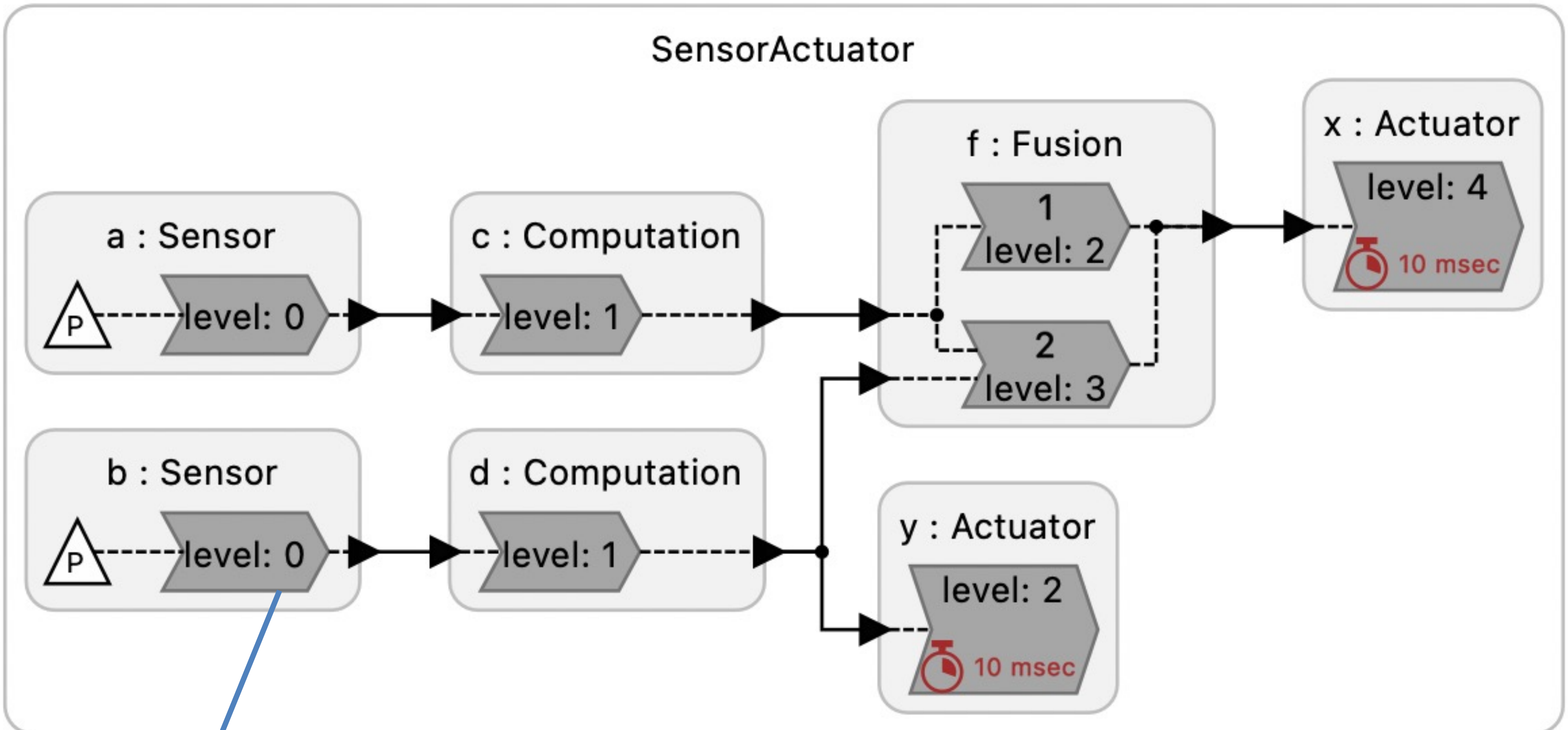
- Sort reactions topologically based on precedences.
- Global notion of "current tag" *g.*
- Event queue containing future events.
- Choose earliest tag *g'* on the event queue.
- Wait for the real-time clock to match the timestamp of *g.*
- Execute reactions sequentially in topological sort order.

Smarter, Parallel Execution



- Sort reactions topologically based on precedences.
- Global notion of “current tag” *g.*
- Event queue containing future events.
- Choose earliest tag *g’* on the event queue.
- Wait for the real-time clock to match the timestamp of *g.*
- **Execute reactions in parallel where possible.**

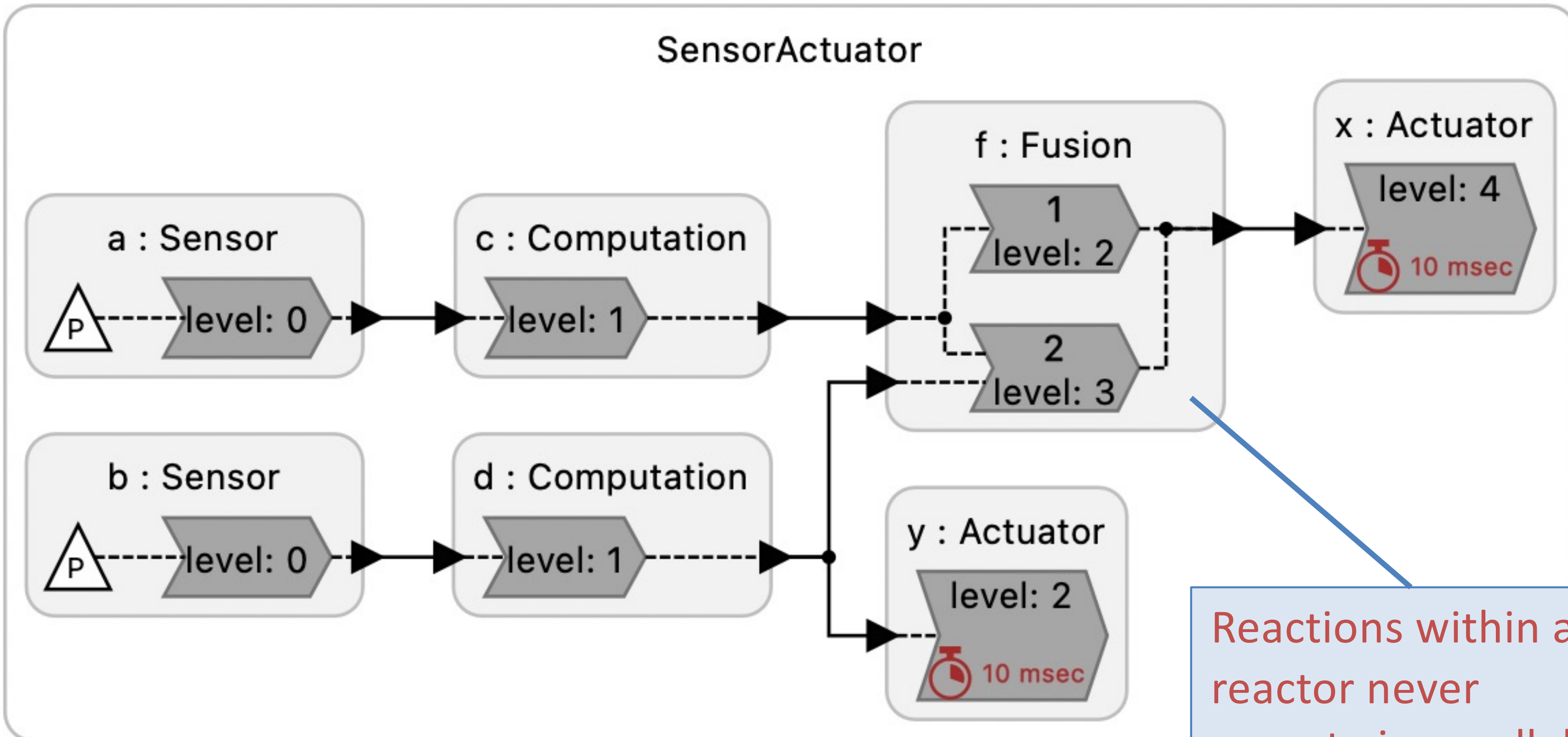
Parallel Execution Using Levels



The level is the depth in a directed acyclic graph of reactions that have dependencies at a tag.

Reactions with the same level can always execute in parallel.

Parallel Execution Using Levels

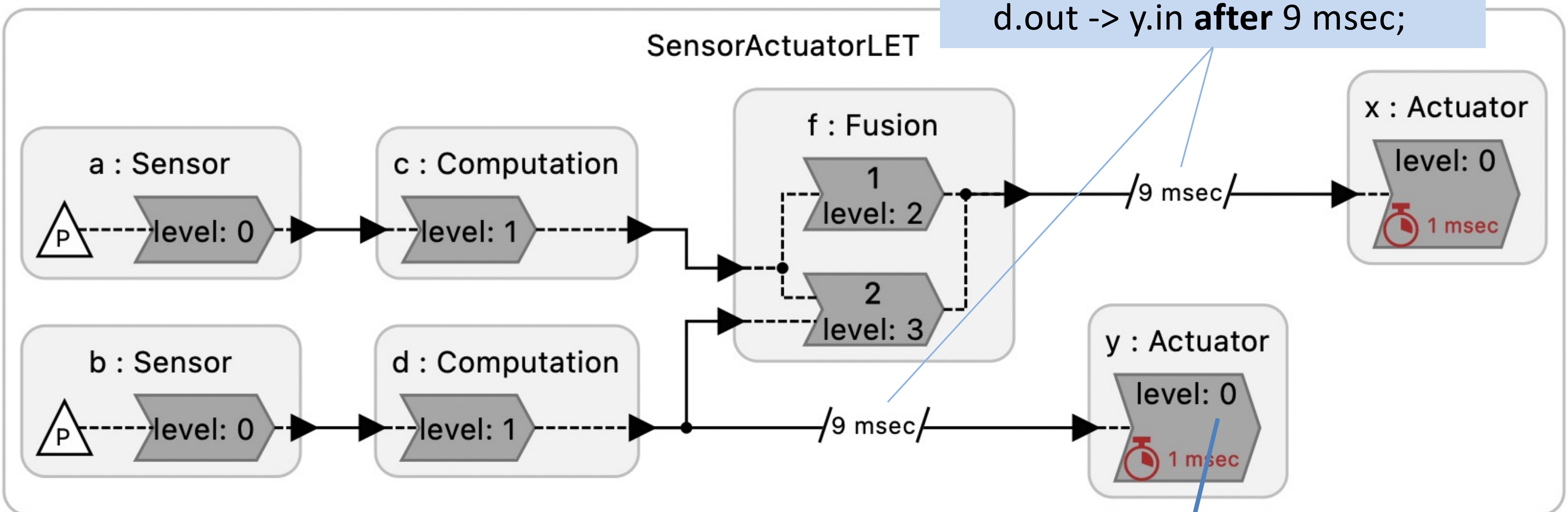


Reactions within a reactor never execute in parallel.

Reactions with the same level can always execute in parallel.

More Deterministic Timing

f.out -> x.in **after** 9 msec;
d.out -> y.in **after** 9 msec;



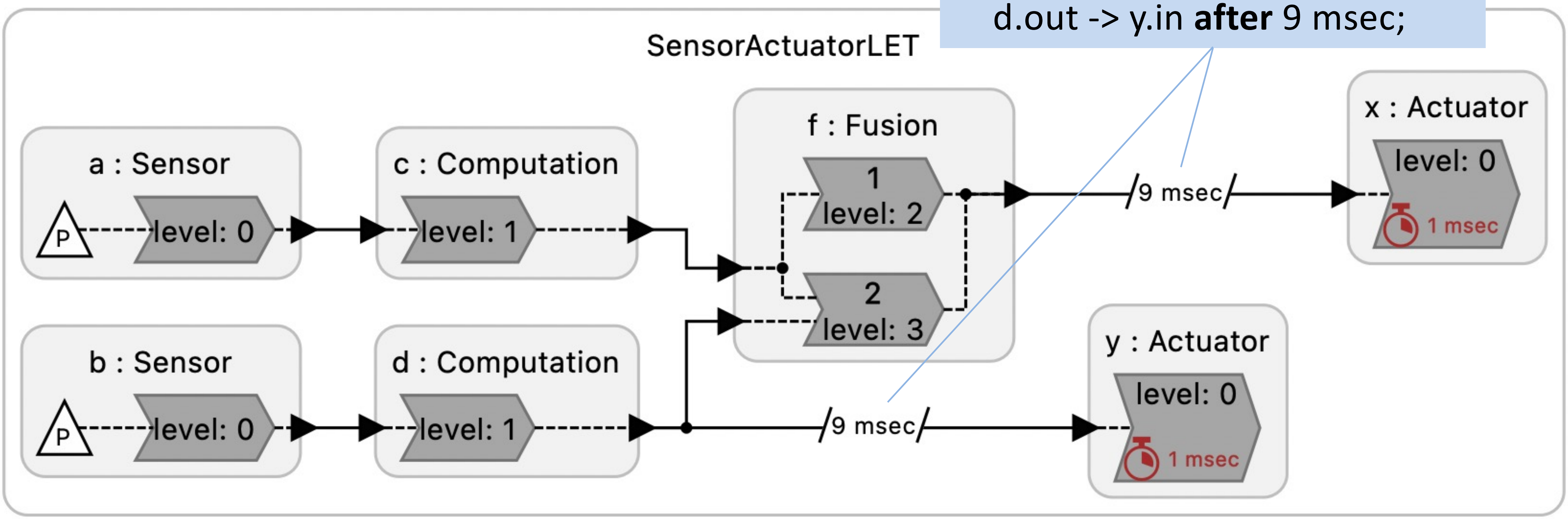
Actuators will now execute between 9 and 10 msec after sensors, unless a deadline violation occurs.

Notice that the level is now 0. Combine with EDF scheduling, and actuator execution has highest priority.

What could cause a deadline violation?

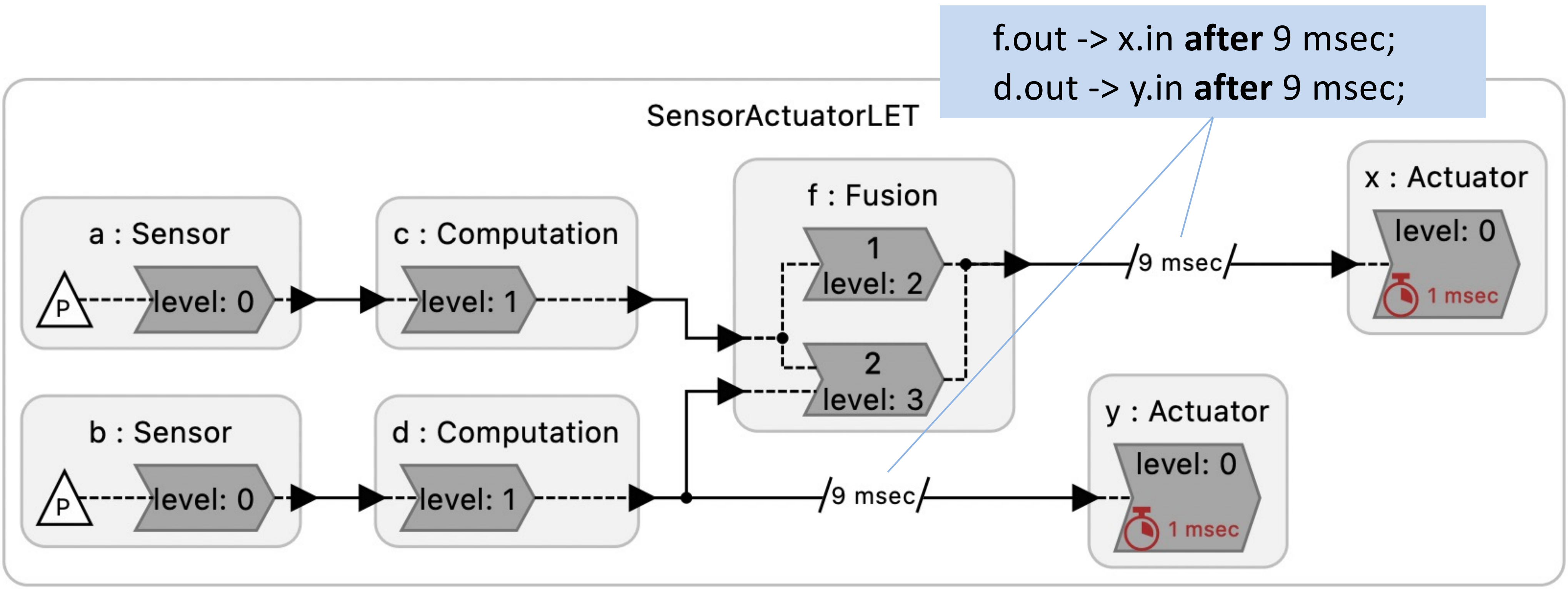
More Deterministic Timing

```
f.out -> x.in after 9 msec;
d.out -> y.in after 9 msec;
```



This strategy is closely related to the notion of **Logical Execution Time** (**LET**) but generalizes that concept to permit zero execution time and to allow deadline violation handlers.

More Deterministic Timing



Classical real-time systems scheduling and execution-time analysis determines whether the specification can be met.

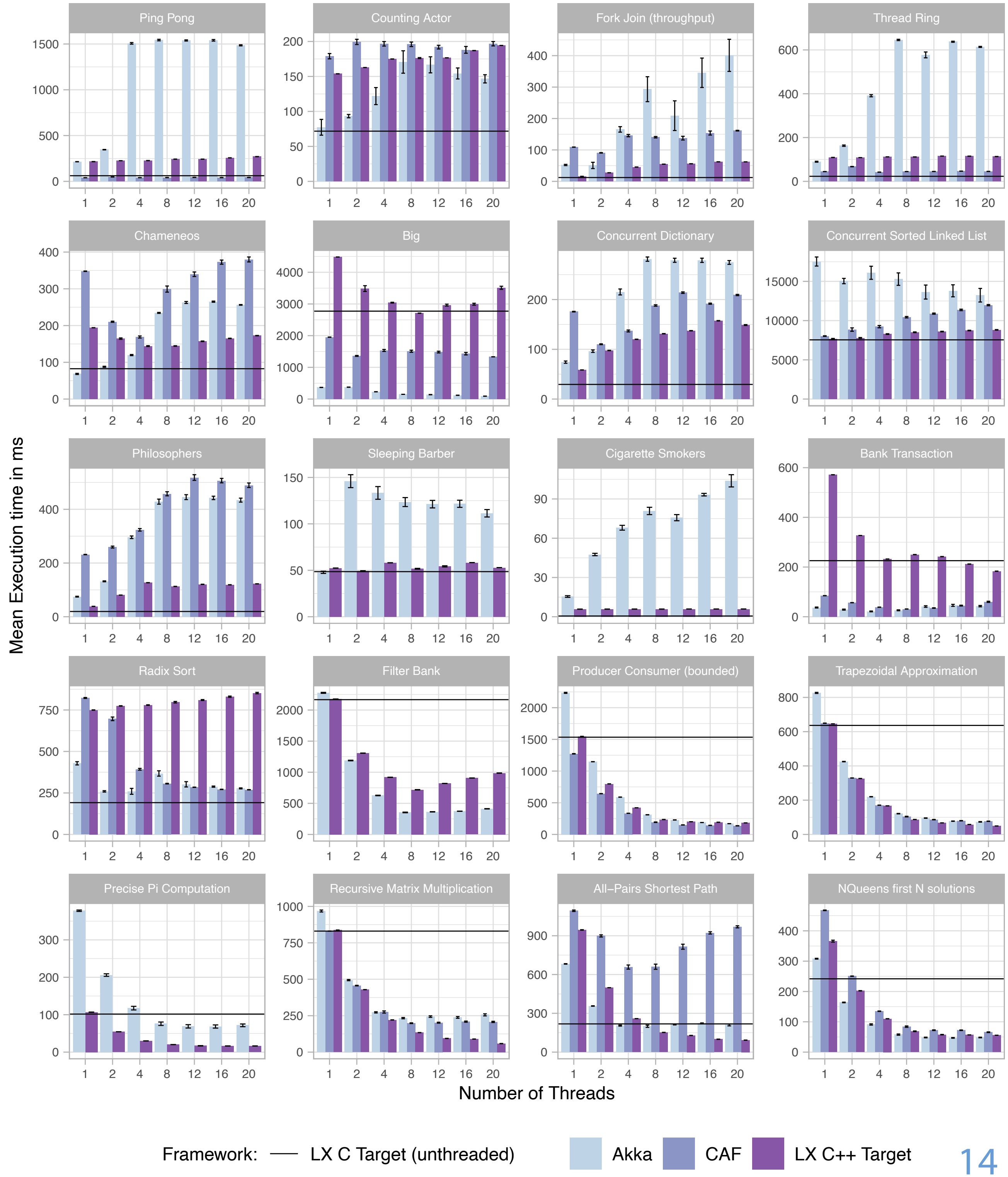
[Buttazzo, 2005] [Wilhelm et al., 2008]

Performance

Determinism does not imply a cost in performance.

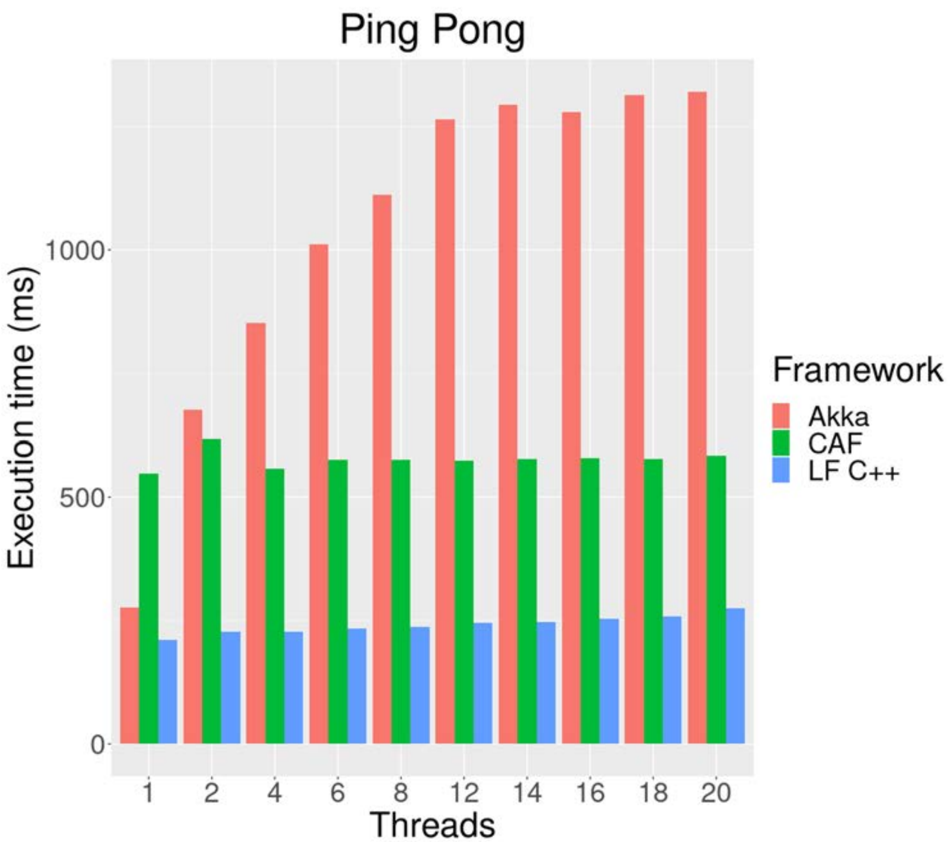
Parallel execution (multicore) does not imply nondeterminism.

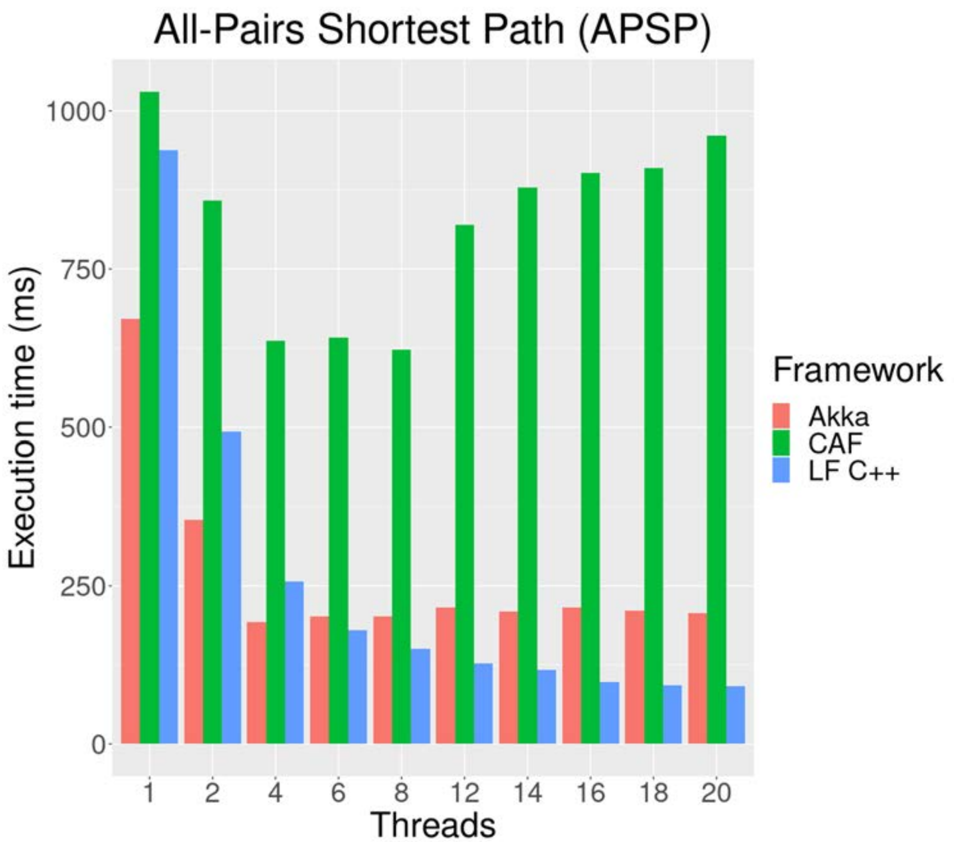
Christian Menard
(TU Dresden)



Scalability

Christian Menard
(TU Dresden)





Active Research

- More aggressive parallel execution.
- Reducing contention for reaction queue.
- Supporting parallel execution at multiple tags.
- Direct support for Logical Execution Time (LET)
- Leveraging lock-free concurrency.