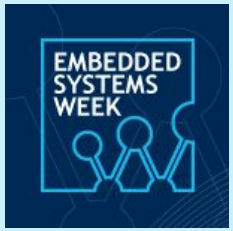


CyPhy '19, October 18 2019, NYC

Reactors: A Deterministic Model for Composable Reactive Systems

Marten Lohstroh, Íñigo Íncer Romeo, Andrés Goens, Patricia Derler,
Jeronimo Castrillon, Edward A. Lee, Alberto Sangiovanni-Vincentelli

Structural Problems in CPS Design

From a embedded software programmer's perspective:

1. Writing deterministic concurrent programs is hard
2. Guaranteeing timing properties is often impossible

Programming Model

Reactors
Model of Computation

- ➔ Timed
- ➔ Synchronous
- ➔ Deterministic
- ➔ Concurrent
- ➔ Event-driven

Establishes an unambigious relationship between physical and logical time.

Lingua Franca
Coordination Language

- ➔ Polyglot
- ➔ Multi-core
- ➔ High-performance
- ➔ Low overhead
- ➔ Compiler + IDE

For specifying deterministic behaviors and imposing timing constraints.

https://github.com/icyphy/lingua-franca

Contributions of our CyPhy Paper

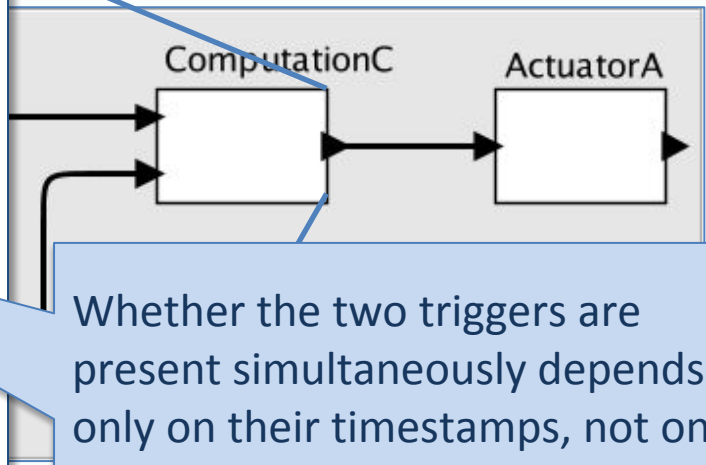
Logical time	t
Physical time	T
Set of tags	$\mathbb{T} = \mathbb{N}^2$
Reactors	
Reactor instance	$r = (I, O, A, S, \mathcal{F}, \mathcal{N}, \mathcal{R}, \mathcal{G}, P, \bullet, \diamond)$
Set of input ports for r	$I(r) \subseteq \Sigma$
Set of output ports for r	$O(r) \subseteq \Sigma$
Set of actions for r	$A(r) \subseteq \Sigma$
Initialization action for r	$\bullet(r) \in A(r)$
Termination action for r	$\diamond(r) \in A(r)$
Set of state identifiers for r	$S(r) \subseteq \Sigma$
Set of transformations contained in r	$\mathcal{F}(r)$
Set of reactions contained in r	$\mathcal{N}(r)$
Set of contained reactors of r	$\mathcal{R}(r)$
Topology of reactors in $\mathcal{R}(r)$	$\mathcal{G}(r) \subseteq \left(\bigcup_{r' \in \mathcal{R}(r)} O(r')\right) \times \left(\bigcup_{r' \in \mathcal{R}(r)} I(r')\right)$
Priority function	$P : \mathcal{N} \cup \mathcal{F} \to \mathcal{P}$
Reactor containing reactor r	$C(r)$
Inputs and outputs	
Input, output instance	$i, o \in \Sigma$
Reactions dependent on $i \in I(r)$	$\mathcal{N}(i) = \{n \in \mathcal{N}(C(i)) \mid i \in D(n)\}$
Reactions antidependent on $o \in O(r)$	$\mathcal{N}(o) = \{n \in \mathcal{N}(C(o)) \mid o \in D^{\vee}(n)\}$

me)

f

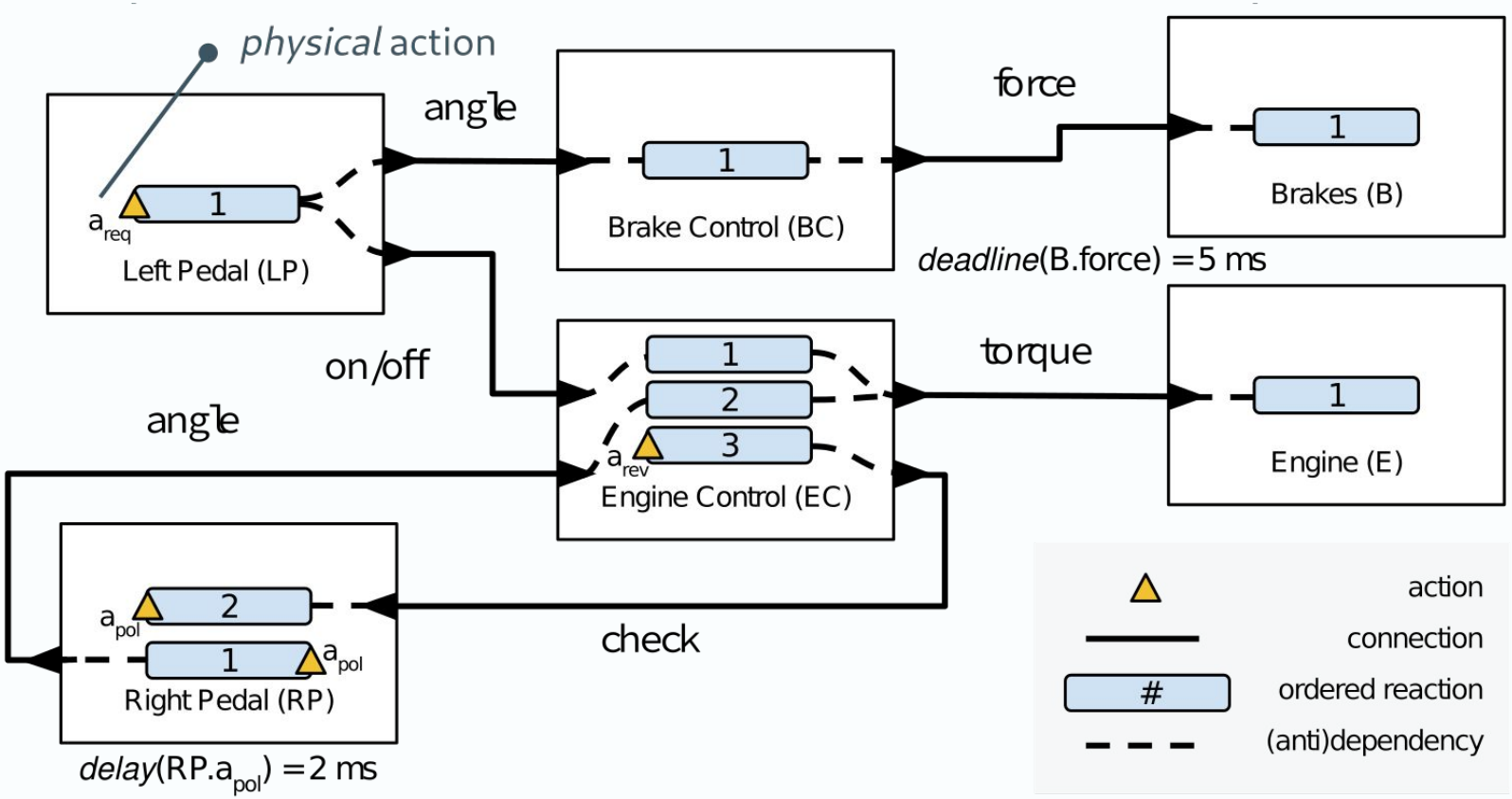
Deterministic Coordination

```
reactor Add {
    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 on the *physical time* of reception nor on where in the network they are sent from.

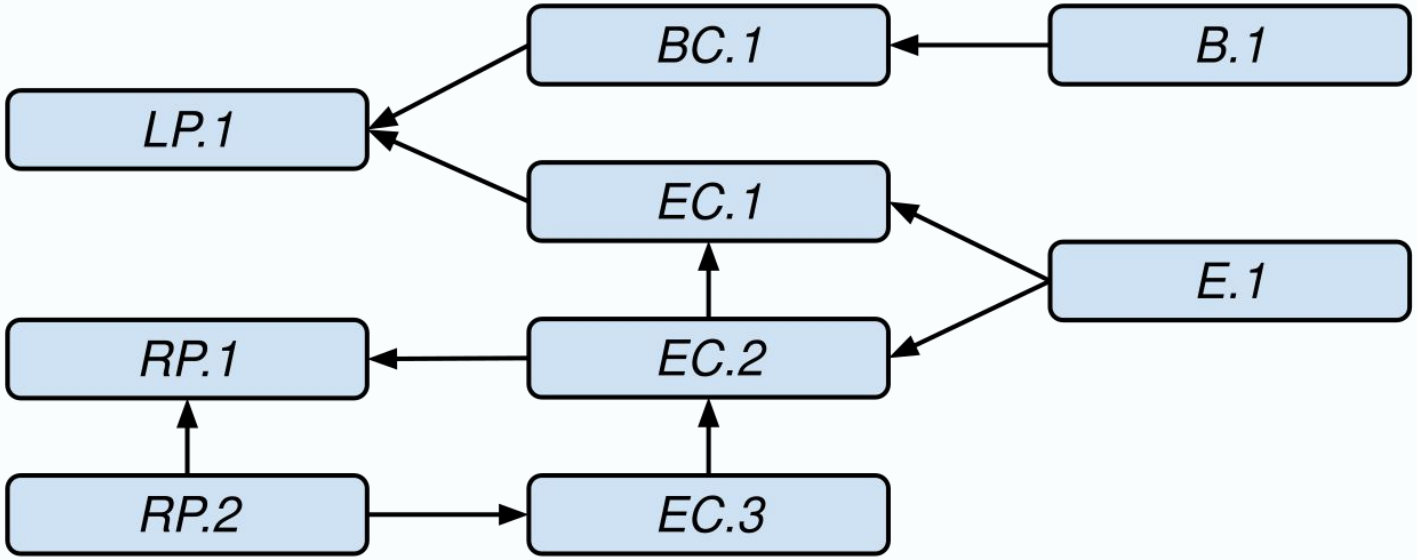
Example: Simplified Power Train



Example: Simplified Power Train

```
target C;

reactor EngineControl {

    input on_off:bool;
    input angle:int;
    output torque:int;
    output check:bool;
    state braking:bool(true);
    timer pulse(0, 10 msec);

    reaction(on_off) -> torque {=
        if(self->braking) {
            self->braking = true;
            set(torque, 0);
        } else {
            self->braking = false;
        }
    =}

    reaction(angle) -> torque {=
        if (!*(self->braking)) {
            set(torque, calc_torq(angle));
        }
    =}

    reaction(pulse) -> check {=
        set(check, true);
    =}
}
```

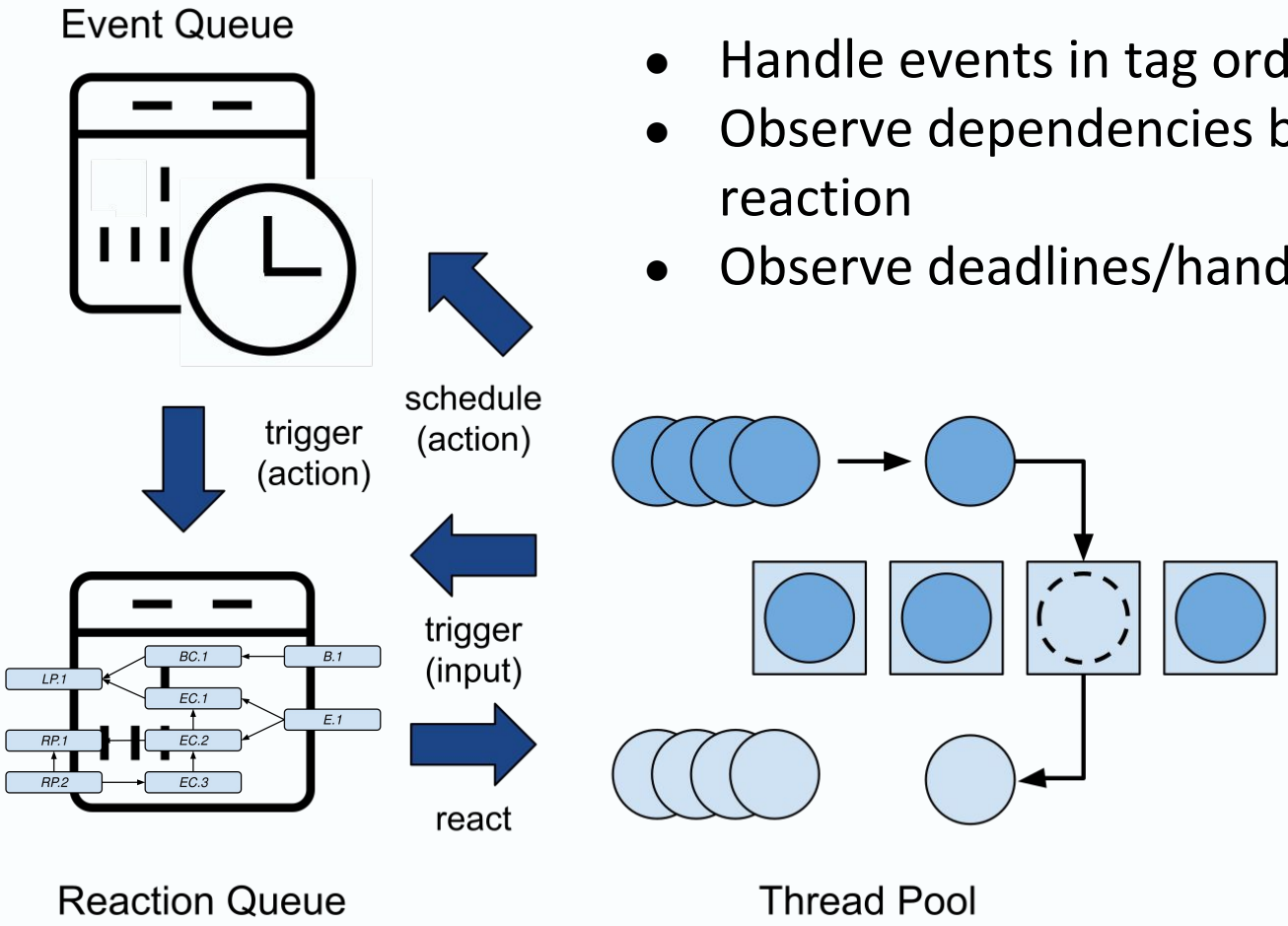
Example: Simplified Power Train



Compilation

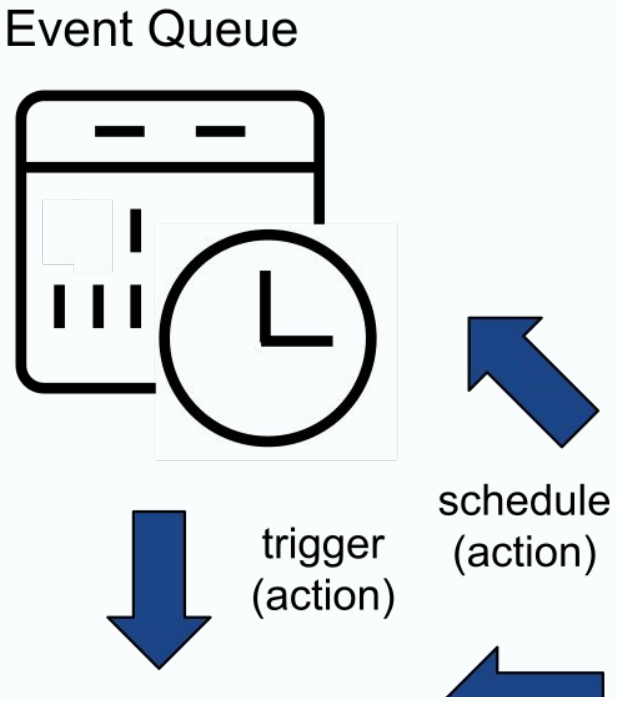
- Parse LF code
  - Exclude reaction bodies written in target code
- Compute dependency graph
  - Reject if cyclic
- Generate code that:
  - Instantiates components
  - Connects ports
  - Incorporates verbatim reaction bodies
  - Brings declared ports and actions into reaction scope
  - Encodes dependencies between reactions
- Invoke the target compiler
  - Report possible compile errors

Runtime: Overview



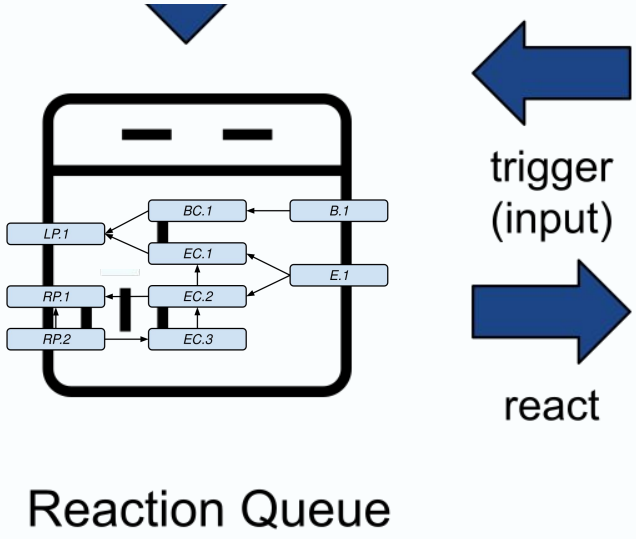
- Handle events in tag order
- Observe dependencies between reaction
- Observe deadlines/handle violations

Runtime: Events



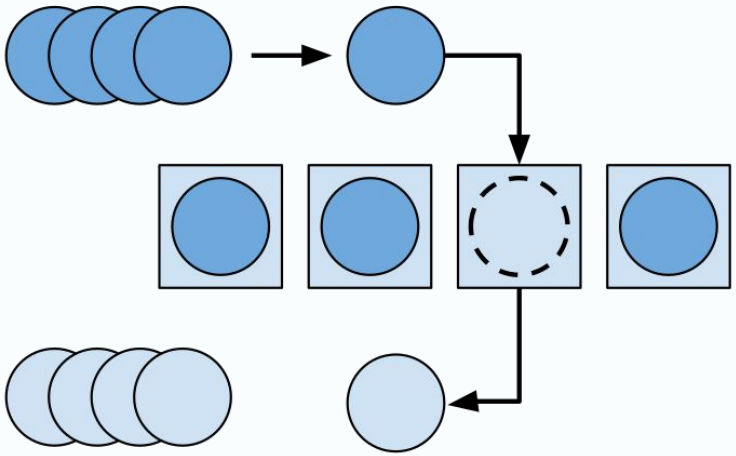
- Events are:
  - associated with an *action* a
  - associated with a *value* v
  - ordered by their *tag* g
- Logical time may advance to the next tag g when g <= T (current physical time)
- The runtime provides a `schedule()` function that can be called from reactions in order to enqueue new events

Runtime: Reactions



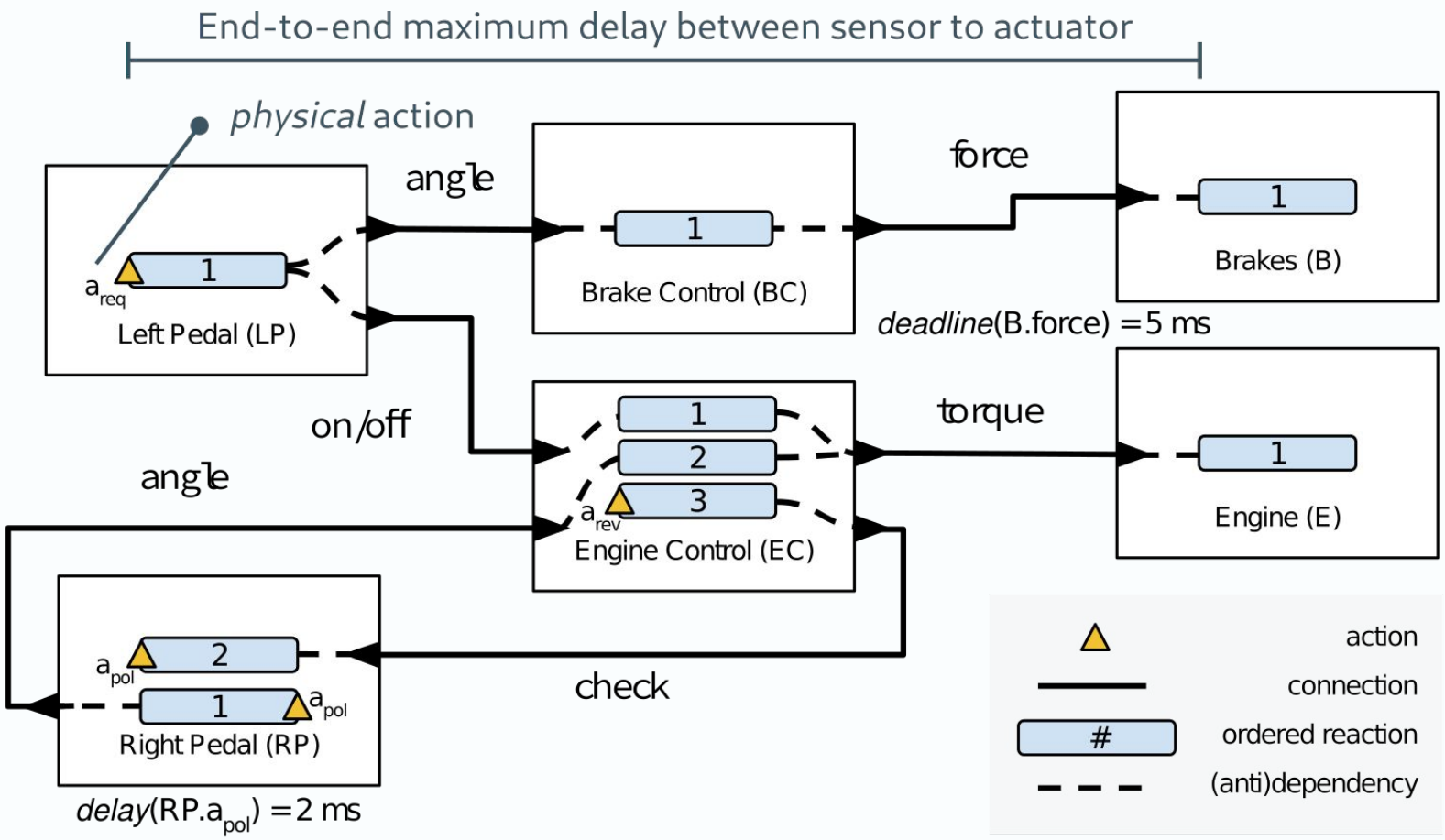
- Reactions are:
  - triggered by actions or inputs
  - able to schedule actions, but only with a tag g > t (current logical time)
- Written outputs trigger instantaneous reactions; these are added *directly* to the reaction queue

Runtime: Workers



- Reactions are allowed to be executed if no reactions they depend on are pending or currently executing
- We exploit concurrency transparently by mapping independent reactions to parallel worker threads
- Only a single mutex is required, and locking only occurs to coordinate between `next()` and `schedule()`

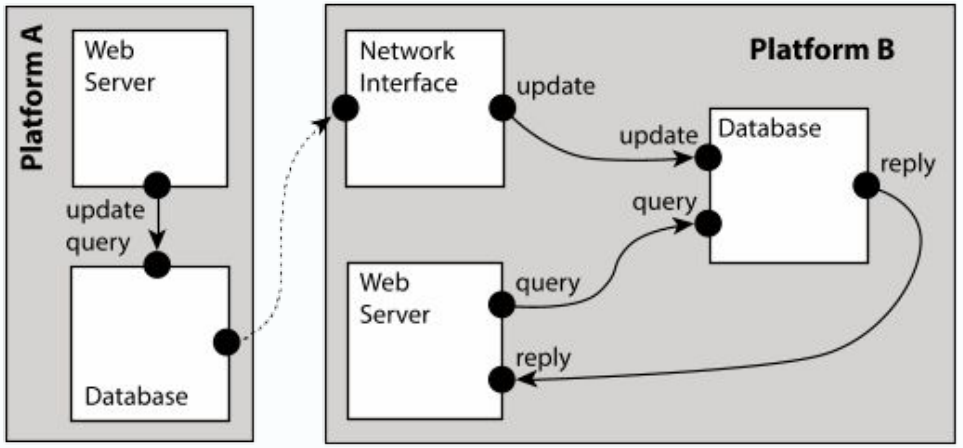
Runtime: Deadlines



Distributed Deterministic Execution

What does the Network Interface reactor do?

- message receipt raises interrupt
- ISR schedules action at t + E + L
- reaction triggered at t + E + L writes to update port



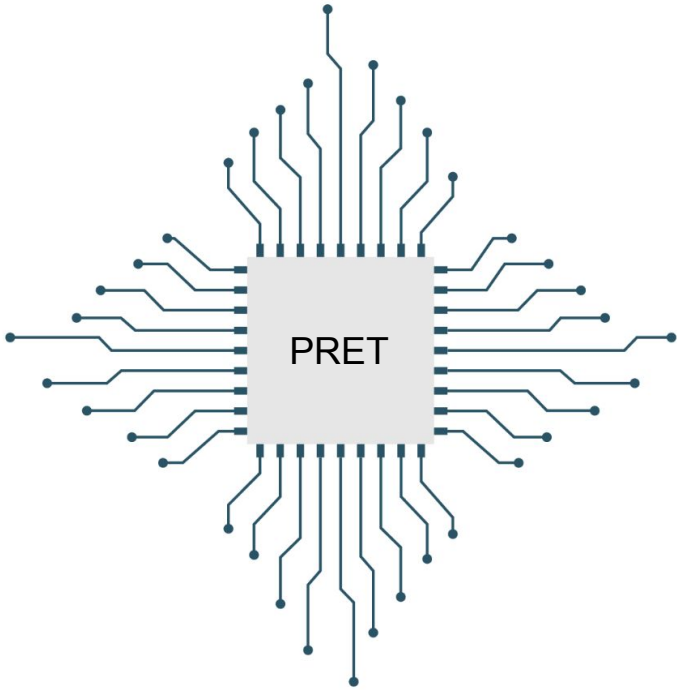
What about the Database?

- reaction triggered by query schedules action at t + E + L
- reaction triggered at t + E + L carries out query, eventually producing reply

Pushing the Envelope

Compile-time Timing Guarantees:

- Timing instructions in the ISA
- WCET performed on reaction bodies
- Feasibility of deadlines determined statically



Relaxing synchronization requirements

***Precomputation**: allow reactions to events with tags greater than the current physical time*

- This can be done without violating the semantics:
  - For reactions that do not depend on physical actions and are part of reactors that are free of observable physical side effects

Next Steps

- Operational Semantics
- Proof of Determinism
- Formalize Deadlines
- Implement mutations
- Modularize scheduler + implement EDF
- More efficient encoding of reaction dependencies
- Target Patmos/PRET
- Develop C++ target, TypeScript target
- Precomputation
- Develop syntax and semantics for higher-order reactor
- Integration with PTIDES
- …

Visit our GitHub:

github.com/icyphy/lingua-franca