T-CREST

Reactors: A Deterministic Model of Concurrent Computation

Martin Schoeberl

Technical University of Denmark

Embedded Systems

- Interact with the world
  - Usually with tight deadlines
  - Real-time systems
- The world is embarrassingly concurrent
  - Best handled with concurrent tasks
- How to represent those tasks?
- How to make a multi-threaded system deterministic?

Multicore Processors

- The path to more performance
- State-of-the art in desktop and server
- Considered for real-time systems
- Provide real (computation) concurrency
  - Communication concurrency?
  - Overlap computation and communication?

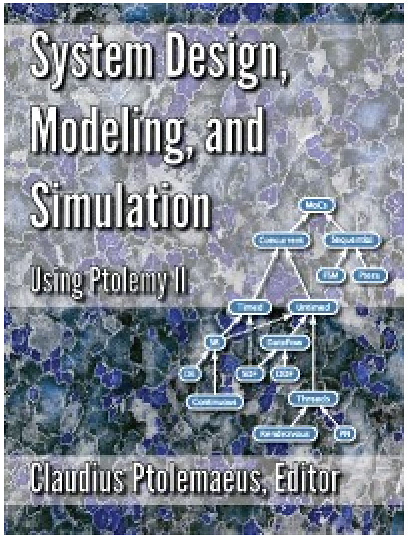
Multicore Usage

- How to program those multiple cores?
  - Use the concurrency for performance
  - Real-time and time-predictable?
- Model of computation as theory

T-CREST

Models of Computation

- MoC is a term used to describe different forms of concurrent execution ‘schemes’
  - See Ptolemy II handbook
  - Explore with Ptolemy II

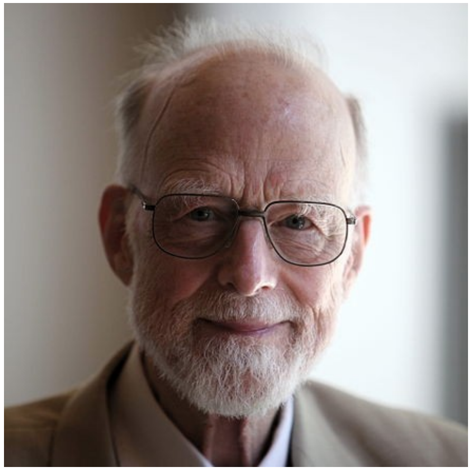


- Processes + communication
- It is about concurrency
- Theoretical models
  - No notion on real-time or performance

Example Models of Computation

- Communicating sequential processes
- Kahn process networks
- Synchronous dataflow
- Actors
- Shared memory and threads
- Reactors

Communicating Sequential Processes (Hoare)



- Sequential processes
- Synchronous message passing
  - Message passing also used for process synchronization
  - Synchronizes with each message
    - No overlap of communication and computation
- Occam as a language for CSPs
  - Hardware implementation in Transputer

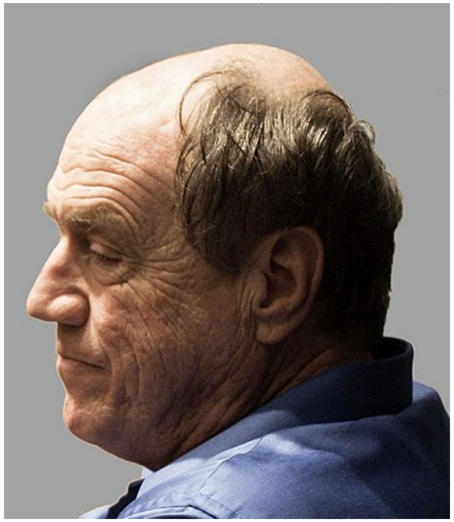
Kahn Process Networks

- Network of processes
  - Also called actors
- Unidirectional channels between processes
  - Unbound buffers
- Process:
  - Reads tokens on input channels
  - Executes the process
  - Produces tokens on the output

Synchronous Data Flow

- Restriction of Kahn process networks
- Fixed number of tokens consumed and produced
  - Can be different for different channels
  - Multi-rate processing
- Balance equations for
  - Number of tokens produced and consumed
  - Execution frequency of actors
- Static schedule for single processor
- Fits well for digital signal processing

Hewitt's Actor Model



- Tasks are represented as actors
- Actors receives messages
- Act (compute) on the reception of a message
- Actors send messages
- Only local/private state
- No need for lock-based synchronization
- Now popular with Akka

Shared Memory and Threads

- Currently most commonly used MoC
- Not well defined
  - Uses threads to represent tasks
  - Uses locks to protect data structures
  - Avoid race conditions
  - Needs a memory model
- Not even mentioned in Ptolemy handbook ;-)

Reactors

- Based on the Actor model
- Add time
  - Original model is untimed
- Add determinism
  - Original order of actions is non-deterministic

Example Reactors

```
reactor Ramp(p:int(10)) {
  input set:int;
  output out:int;
  clock c(p);
  constructor {=
    int count = 0;
  =}
  reaction(c) -> out {=
    count++;
    set(out, count);
  =}
  reaction(set) {=
    count = set;
  =}
}
```

```
reactor Print {
  input in:int;
  reaction(in) {=
    printf("%d\n", in);
  =}
}

composite App {
  a = new Ramp(p=100);
  b = new Print();
  a.out -> b.in;
}
```

Reactor Model

- Components contain
  - Input ports, actions, and clocks, all of which are triggers
  - Output ports, local state, and an ordered list of reactions
- Composition
  - A reactor may contain other reactors
  - Connections define the flow of messages
  - An output port may be connected to multiple input ports

Reactor Model

- Events
  - Messages sent from one reactor to another, and clock and action events
  - Each have a timestamp, a value on a logical timeline
  - Each port, clock, and action can have at most one such event at any logical time
  - An event may carry a value that will be passed as an argument to triggered reactions

Reactor Model

- Reactions
  - A procedure in a target language that is invoked in response to a trigger event.
  - A reaction can read input ports and can produce outputs
  - Declare all inputs that it may read and output ports to which it may write
  - All inputs that it reads and outputs that it produces bear the same timestamp as its triggering event
    - The reaction itself is logically instantaneous, so any output events it produces are logically simultaneous with the triggering event

Reactor Model

- Flow of Time
  - Successive invocations of any single reaction occur at strictly increasing logical times
  - Any messages that are not read by a reaction triggered at the timestamp of the message are lost
  - Logical time can be advanced by a delay statement

Reactor Model

- Mutual Exclusion
  - The execution of any two reactions of a reactor are mutually exclusive
    - Atomic with respect to one another
  - Any two reactions that are invoked at the same logical time are invoked in the order specified by the reactor definition
  - Avoiding race conditions between reactions accessing the reactor state variables
    - There is only local state

Time, Delays, and Deadlines

- For real-time systems
- Reactors have two notions of time
  - Logical time
  - Physical time
- Each event has a timestamp
  - Sensor input uses physical time for the timestamp
  - Otherwise, this is logical time

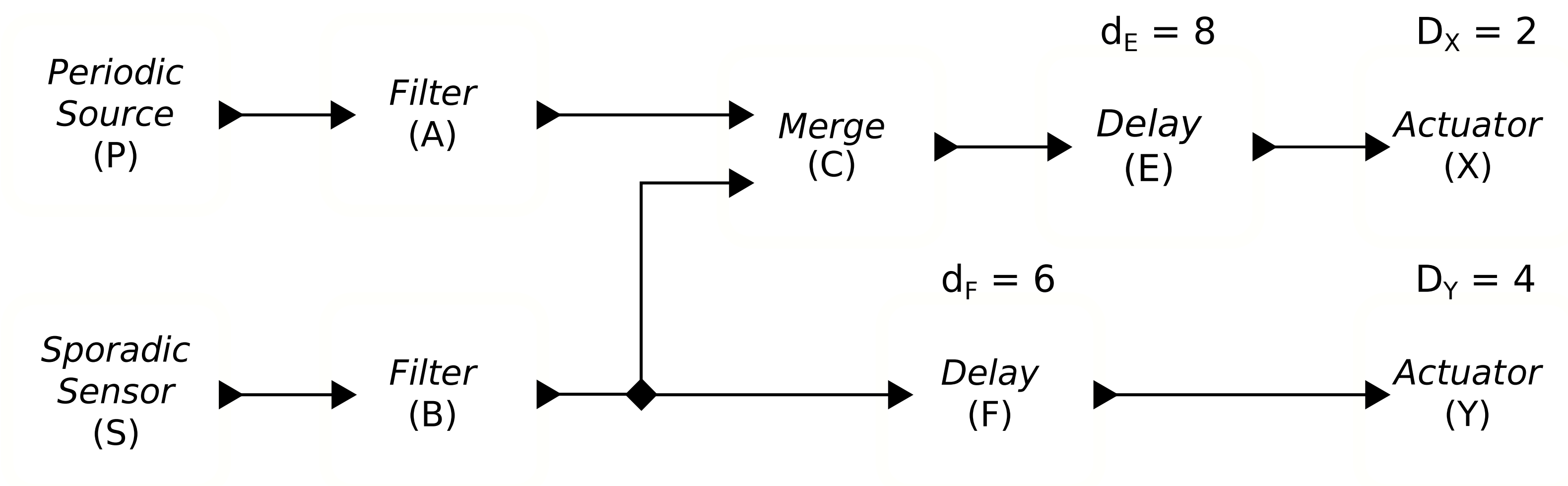
Time, Delays, and Deadlines

- Multiple events with the same timestamp are simultaneous
- Logical time does not advance when executing an action
- Logical time and physical time are aligned at application start
- Logical time never gets ahead of physical time

Time, Delays, and Deadlines

- Inputs are assigned a timestamp of physical time
- Logical time may lag behind physical time
- Deadlines at actuators to ensure real-time behavior
- Delay construct to align logical time to physical time

Example



Periodic Source (P)
Filter (A)
Merge (C)
$d_E = 8$
Delay (E)
$D_X = 2$
Actuator (X)
Sporadic Sensor (S)
Filter (B)
$d_F = 6$
Delay (F)
$D_Y = 4$
Actuator (Y)

Logical Time and Physical Time

<u>Logical Time</u>

<u>Physical Time</u>

- Steps or 'ticks'
- Discrete
- Absolute Simultaneity

- External events
- Deadlines
- Fault handling

- Measurements
- Continuous
- Relativistic Simultaneity

The Persistence of Memory — Salvador Dalí

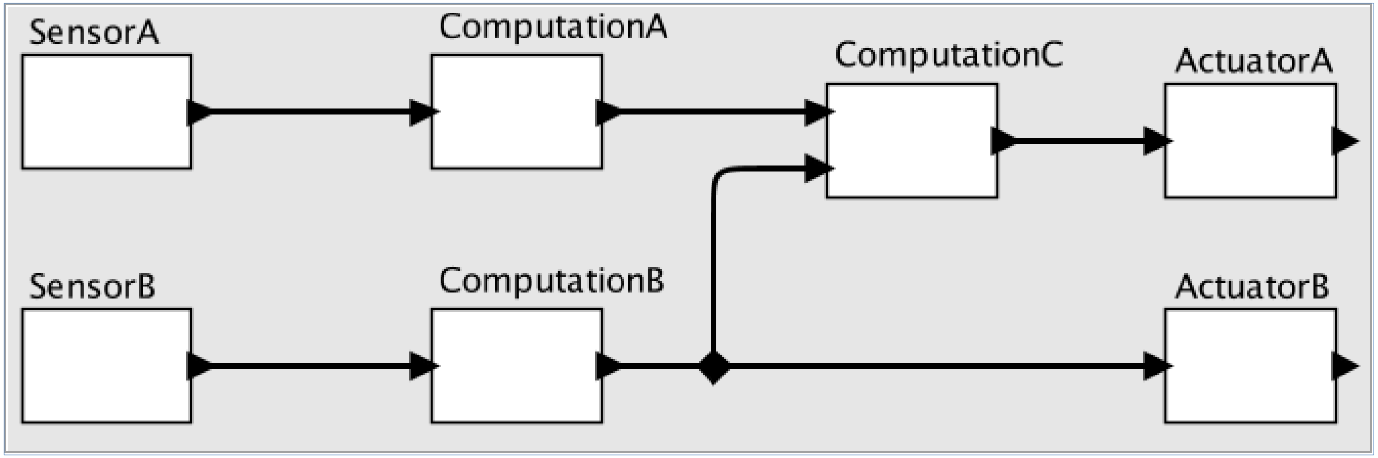
Lingua Franca (LF)

- A polyglot meta-language for deterministic, concurrent, time-sensitive systems
- Reaction code in target language
  - C, C++,…
- Generation of runtime code from LF constructs

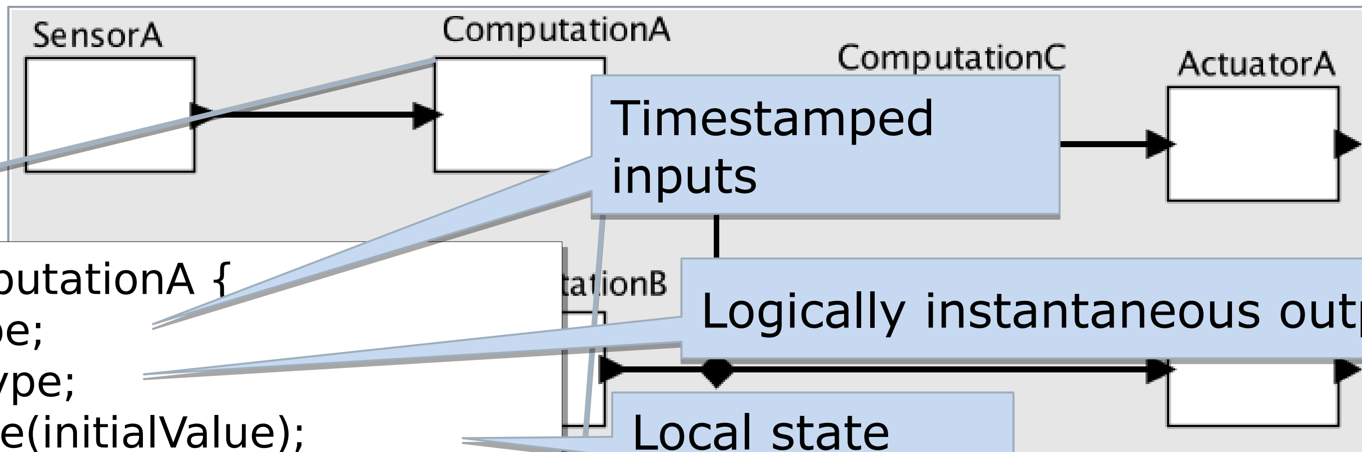
Hierarchical Composition and Ports

```
reactor A {
  output y;
  ...
}
reactor B {
  input x;
  ...
}
main reactor C {
  a = new A();
  b = new B();
  a.y -> b.x;
}
```

Application Sketch



Reactors




```
reactor ComputationA {
    input x:type;
    output y:type;
    state s:type(initialValue);
    reaction(x) -> y {=
        Target-language code
        referencing x, y, and s.
    =}
}
```

Timestamped inputs

Logically instantaneous outputs

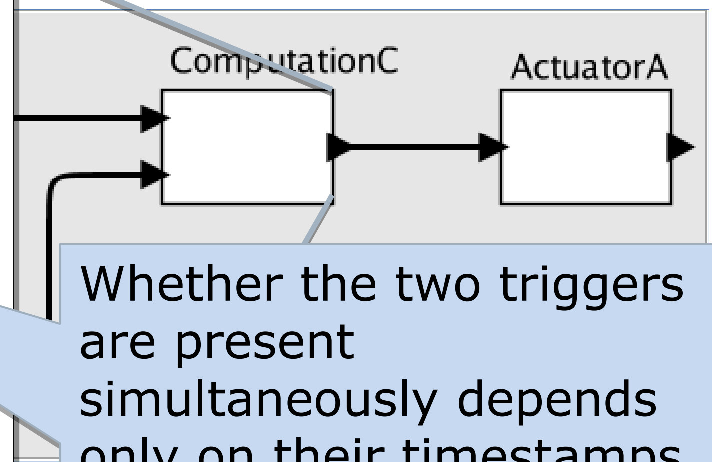
Local state

Reaction signature gives trigger(s) and production

Application logic given in a target language (C, C++, Python, TypeScript, …)

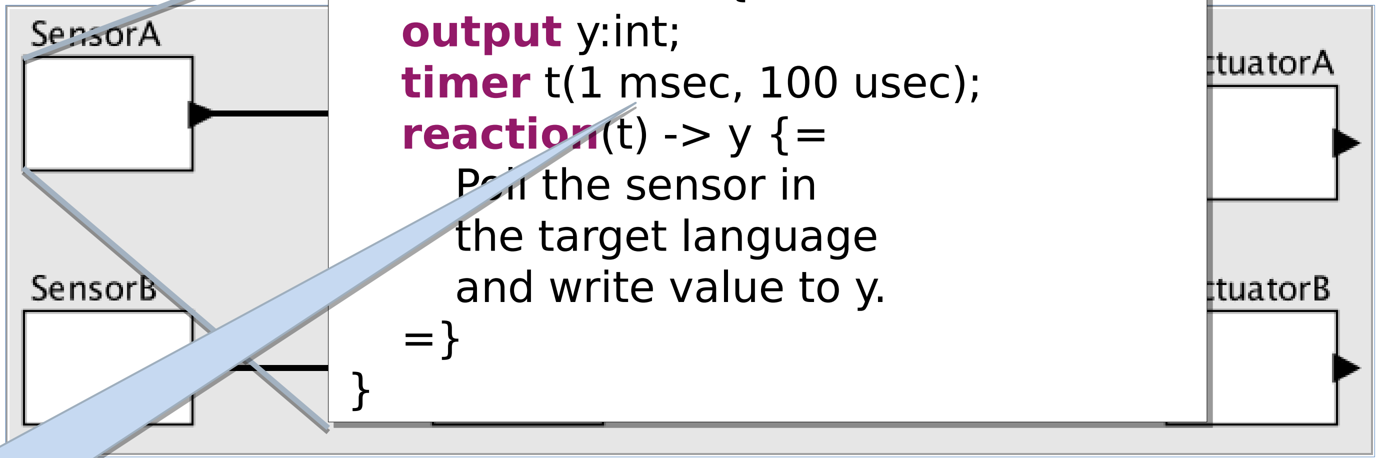
Determinism

```
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 when they are received nor on where in the network they are sent from.

Periodic Behavior

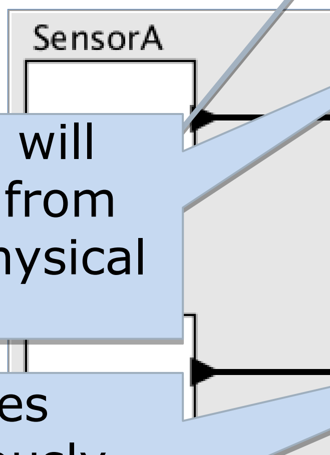



```
reactor SensorA {
    output y:int;
    timer t(1 msec, 100 usec);
    reaction(t) -> y {=
        Poll the sensor in
        the target language
        and write value to y.
    =}
}
```

Time as a first-class data type.

In our C target, timestamps are 64-bit integers representing the number of nanoseconds since Jan. 1, 1970 (if the platform has a clock) or the number of nanoseconds since starting (if not).

Event-Triggered Behavior



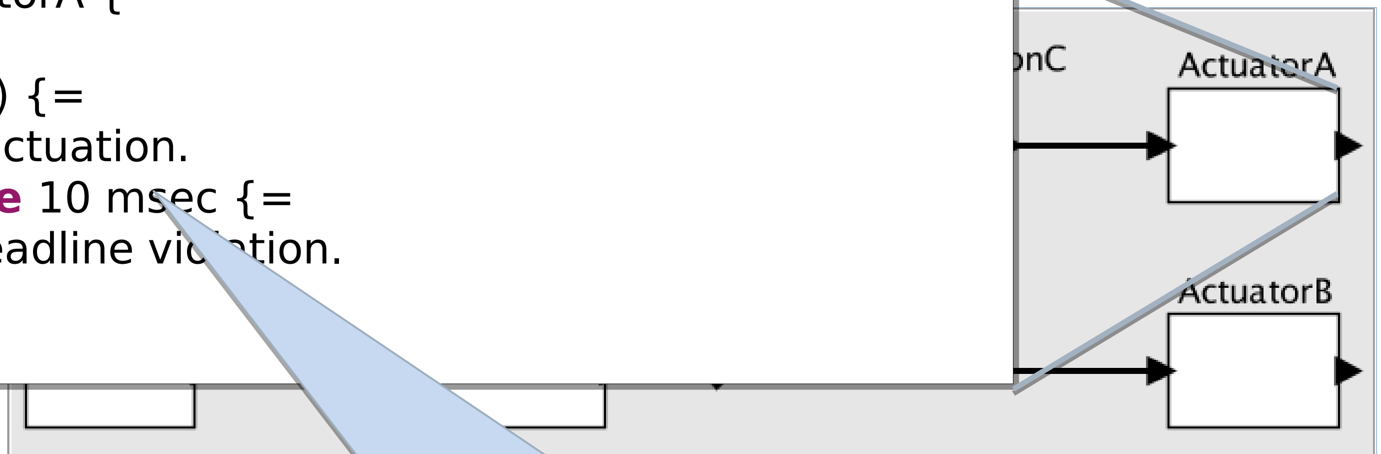
Timestamp will be derived from the local physical clock.

ISR executes asynchronously, and schedule() function is thread safe.

```
reactor SensorB {
    output y:int;
    physical action a:int;
    timer start;
    reaction(startup) -> a {=
        Set up an interrupt service
        routine that will call:
        schedule(a, 0, value);
    =}
    reaction(a) -> y {=
        set(y, a_value);
    =}
}
```

Deadlines

```
reactor ActuatorA {
    input in:int;
    reaction(in) {=
        perform actuation.
    =} deadline 10 msec {=
        handle deadline violation.
    =}
}
```



Deadline is violated if the input d.x triggers more than 10 msec (in physical time) after the timestamp of the input.

Lingua Franca Tooling

ReflexGame.lf

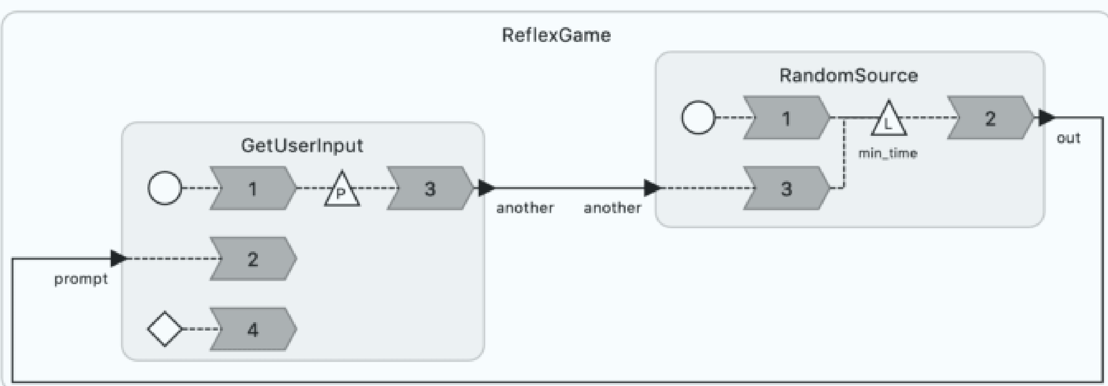
```
    reaction(shutdown) {=
        if (self->count > 0) {
            printf("\n**** Average response time: %d.\n", self->
        } else {
            printf("\n**** No attempts.\n");
        }
    =}
}
main reactor ReflexGame {
    p = new RandomSource();
    g = new GetUserInput();
    p.out ->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

```

Verbatim target code

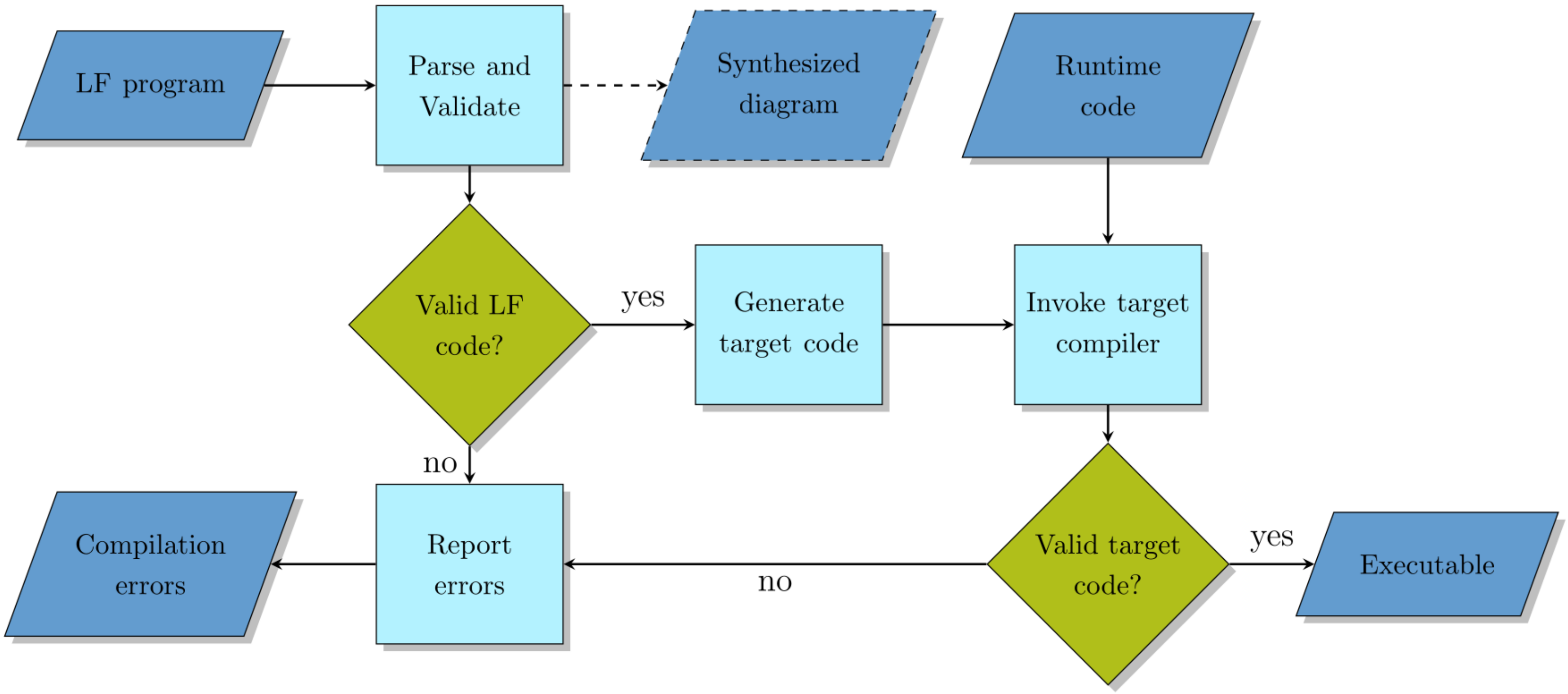
Reactor-oriented composition layer

Tasks Console Error Log Diagram



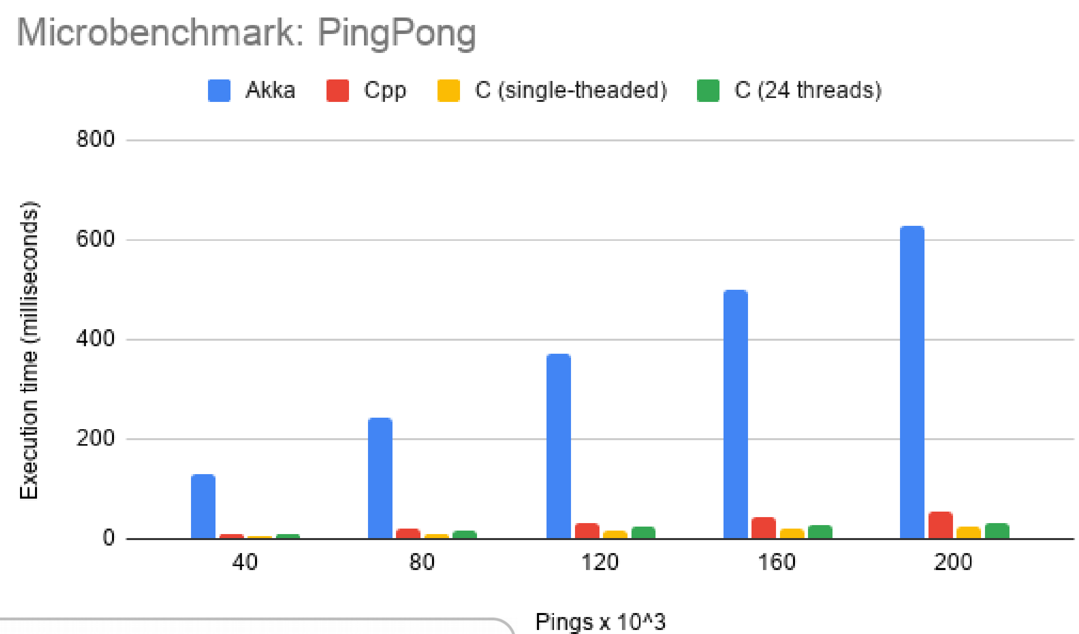
Interactive Visualization

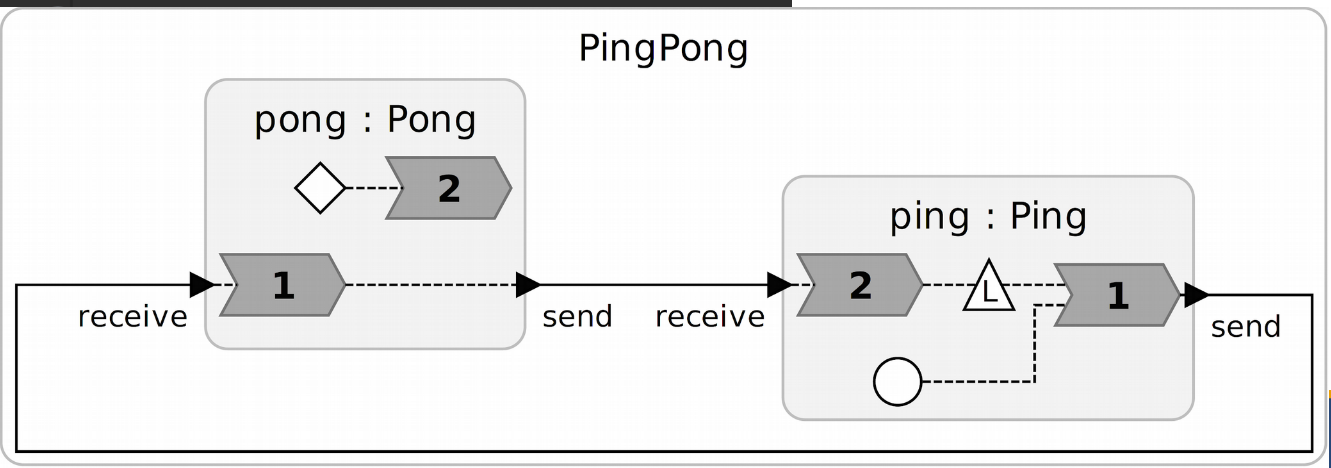
LF Compiler Toolchain



Runtime Overhead

```
reactor Ping(count:int(1000000)) {
    input receive:int;
    output send:int;
    state pingsLeft:int(count);
    logical action serve;
    reaction (startup, serve) -> send
        SET(send, self->pingsLeft--);
    =}
    reaction (receive) -> serve {=
        if (self->pingsLeft > 0) {
            schedule(serve, 0);
        } else {
            request_stop();
        }
    =}
}
```





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.

LF Status

Still early, but evolving rapidly

- Eclipse/Xtext-based IDE
- C, C++, Python, and TypeScript targets
- Code runs on Mac, Linux, Windows, and Patmos (T-CREST)
- EDF scheduling on multicore
- Command-line compiler
- Regression test suite
- Wiki documentation

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

MoC/Reactors Implementation

- No notion about implementation
  - On a multicore with a network-on-chip?
- All MoCs can be (and have been) implemented with:
  - Shared memory and
  - Locks
- Maybe better with hardware support?
  - E.g., CSP had HW support with Transputers

Real-Time Systems

- Need to know the worst-case execution time (WCET)
- T-CREST is built to enable WCET analysis
- A time-predictable platform for reactors
- Multicore for higher performance

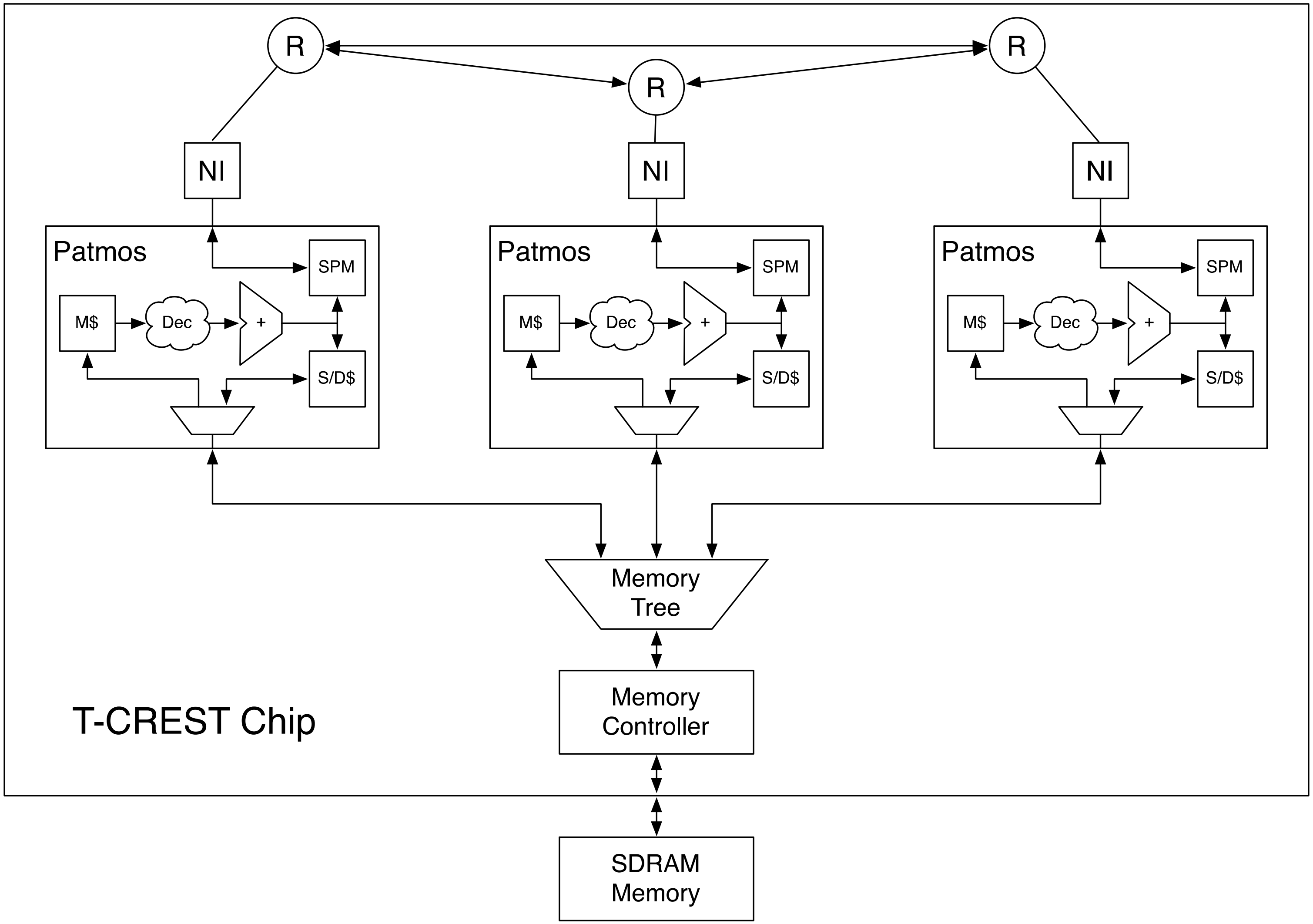
T-CREST Multicore Processor

- Time-predictable platform
- Patmos dual-issue processor
  - Special time-predictable caches
- Argo and S4NOC network-on-chip
  - Hardware for message passing
  - Time-division multiplexing of channels and routers
- Time-predictable SDRAM memory controller

T-CREST cont.

- Main memory access tree
  - Time-division multiplexing
- Compiler based on LLVM
  - Support of Patmos
  - Single-path code generation
- WCET Analysis tools
  - AbsInt aiT, platin, Heptane
- T-CREST availability
  - Open-source on GitHub

T-CREST Hardware



Models of Communication

- How do we move data (bits) around?
- What can we provide in hardware?
- Keep bits on-chip
  - As much as possible
- Have static arbitration for time predictability
  - Use time division multiplexing (TDM) for all arbitrations
    - Memory and network-on-chip (NoC)

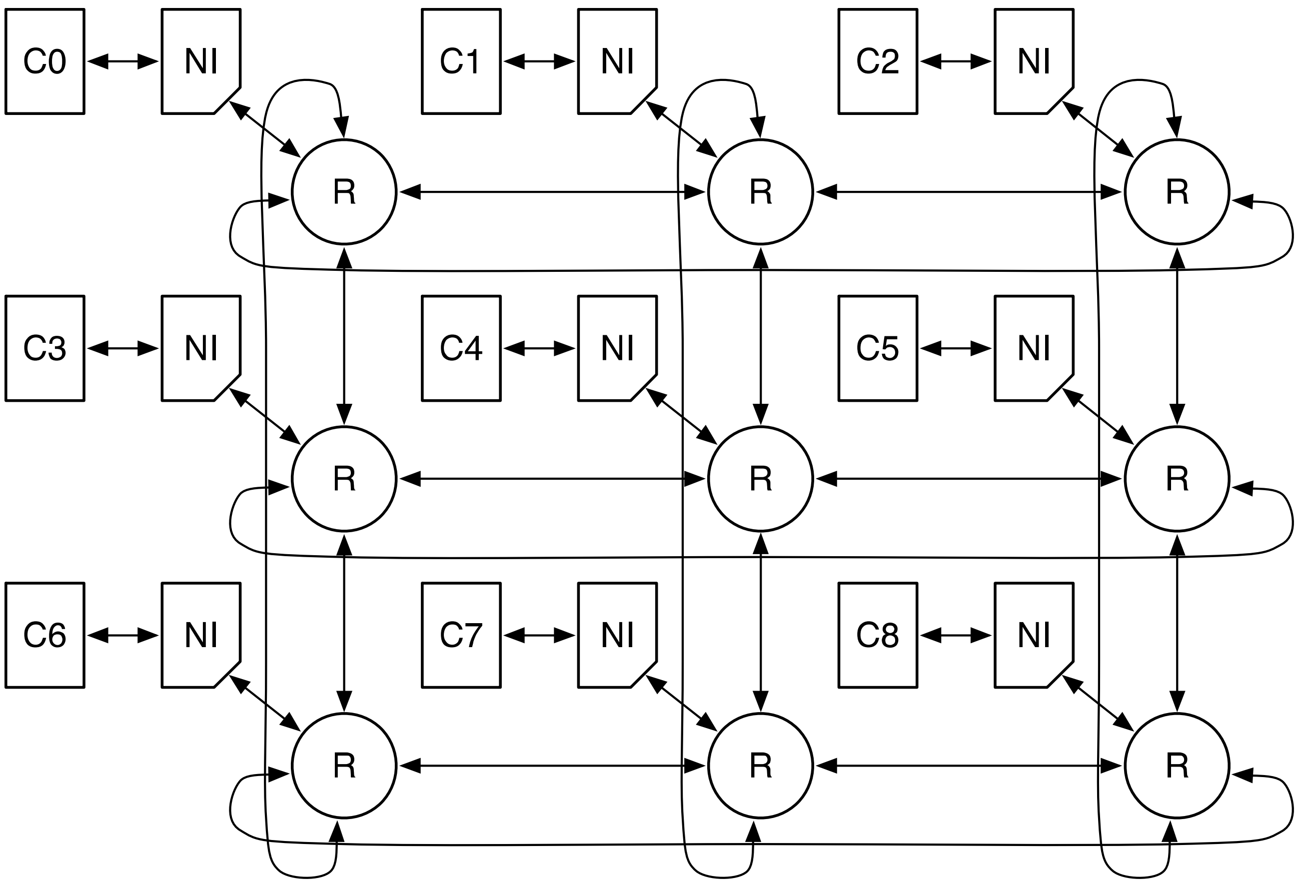
Models of Communication

- Shared main memory
- Network-on-chip (variations)
- Shared scratchpad memory
- Distributed shared on-chip memory
- Shared memory with ownership
- Memory between cores
- One-way shared memory

Shared Memory

- Current form for multicore processors
- L2 cache and cache coherence does the communication
- Memory model needed for visibility rules
  - When is an update of core A visible for core B
- Does not scale well
  - Cache coherence is a many to many communication

Network-on-Chip (NoC)



Network-no-Chip

- Push communication
  - Simple and most efficient
  - Implemented in T-CREST Argo NoC
- Push and pull
  - Pull needs about double the bandwidth
- Balance between on-chip memories and NoC to be explored
- Probably a good fit for reactor events

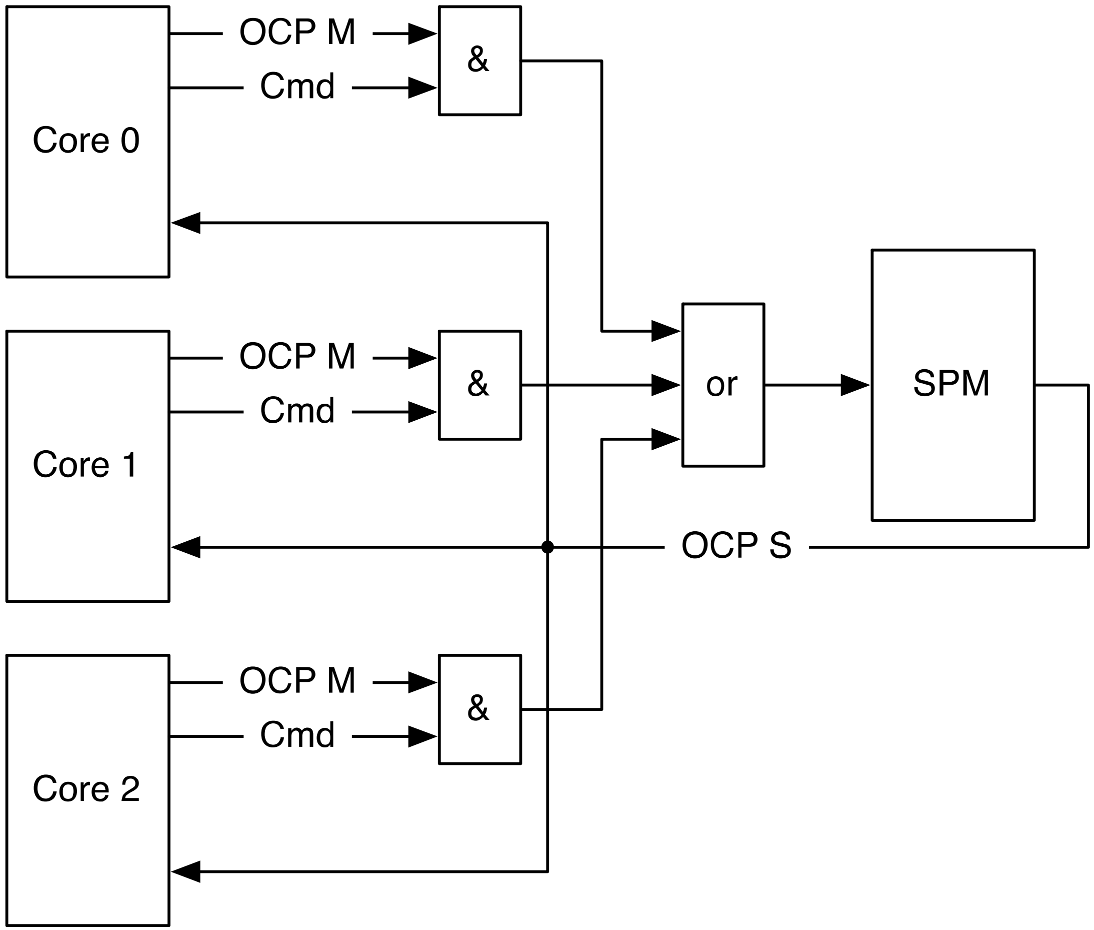
Shared Scratchpad Memory

- Core local SPMs are popular
  - On-chip memory
  - Instead of caches
  - Time predictable
- Shared SPM instead of L2 cache
- All cores access with TDM arbiter
  - Time-predictable
  - Shorter access time than to shared external memory

Shared SPM with Ownership

- A SPM that is *owned* by a core
- Ownership can be transferred
- Only one core can access when owning it
  - Shorter access time than shared SPM
- Good for bulk data movement between cores
- We can have a pool of shared SPMs

SPM with Ownership



Distributed Shared Memory

- Combine local SPMs with a NoC
  - Two NoCs: write/cmd and read return
- Accessible from all cores
  - Via a network-on-chip
  - Different access times
- Dist. shared memory vs. shared SPM
  - Some programming model
  - Both keep data on-chip
  - Dist. shared memory + NoC higher BW than shared SPM

Hardware for Message Passing

- All presented architectures can be used for message passing
- Add HW for more efficient handling
  - Special instructions for send and receive
    - RISC-V extended for the Paternoster NoC
  - Argo NoC has DMAs that handle concurrent messages transmission
  - Hardware support for CSP rendezvous

Reactor Projects (Ongoing Work)

- LF for Patmos available (multicore)
- Current LF implementation uses shared memory for events
- Explore multicore communication for the messages
- Do WCET analysis of reactors
- Could be a MS thesis project

Conclusion

- Models of computation are well defined
  - But seldom used
  - The communication mechanics is not considered
- Lingua Franca programs are testable (timestamped inputs -> timestamped outputs)
- LF programs are deterministic under clearly stated assumptions
- Violations of assumptions are detectable at run time

Visit LF on GitHub

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