iCyPhy

Taking Advantage of Clock Synchronization in Distributed Computing

Edward A. Lee

Professor of the Graduate School
Distinguished Professor Emeritus

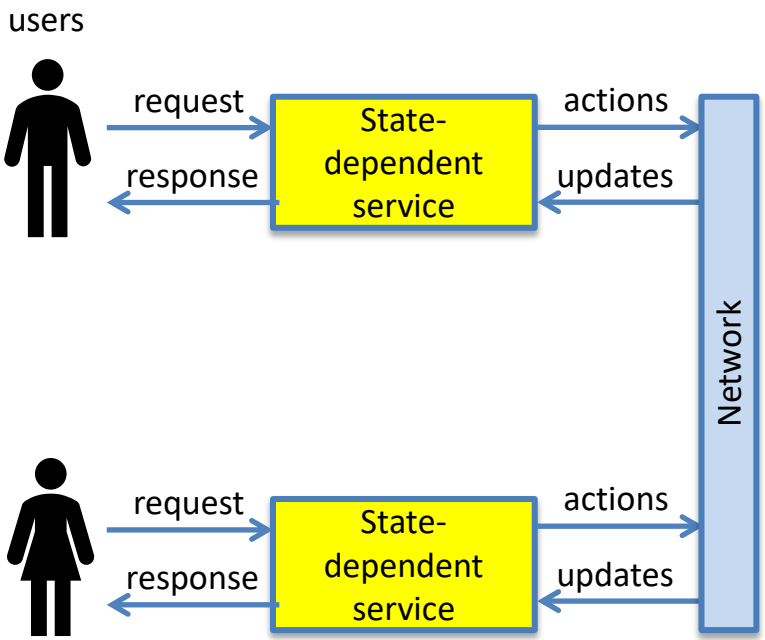
Invited Keynote Talk

Workshop on Synchronization and Timing Systems (WSTS)

May 4-7, Bellevue, Washington

University of California at Berkeley

Distributed System Challenges



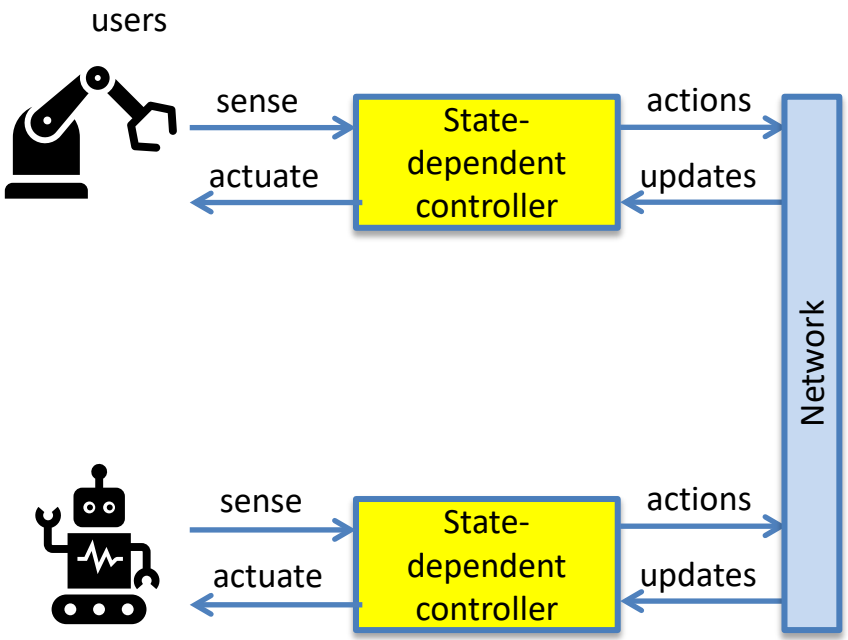
Examples

- Distributed banking
- Financial exchanges
- Auction systems
- Distributed logging

Requirements:

- Consistent balance
- Fair transactions
- Exactly one winner
- Agreement on ordering

Cyber-Physical Examples



Examples

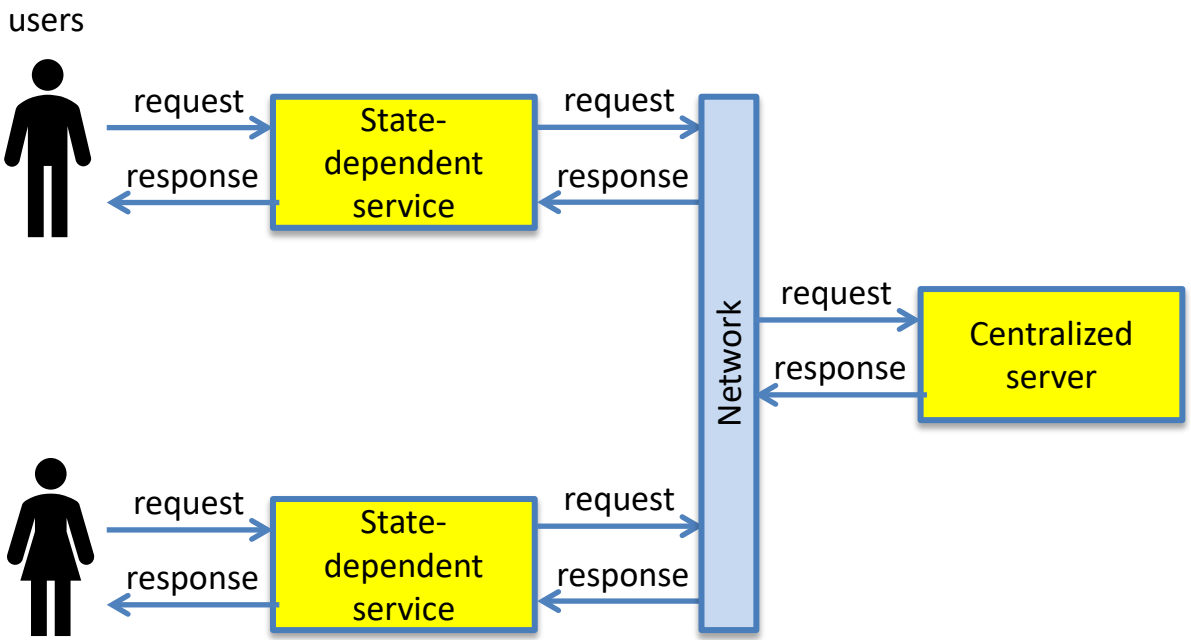
- Factory floor
- Traffic control
- Power systems

Requires a *consistent* view of system state

Requirements:

- Mutual exclusion
- Safe resource sharing
- Operating limits

Classic Centralized Solution



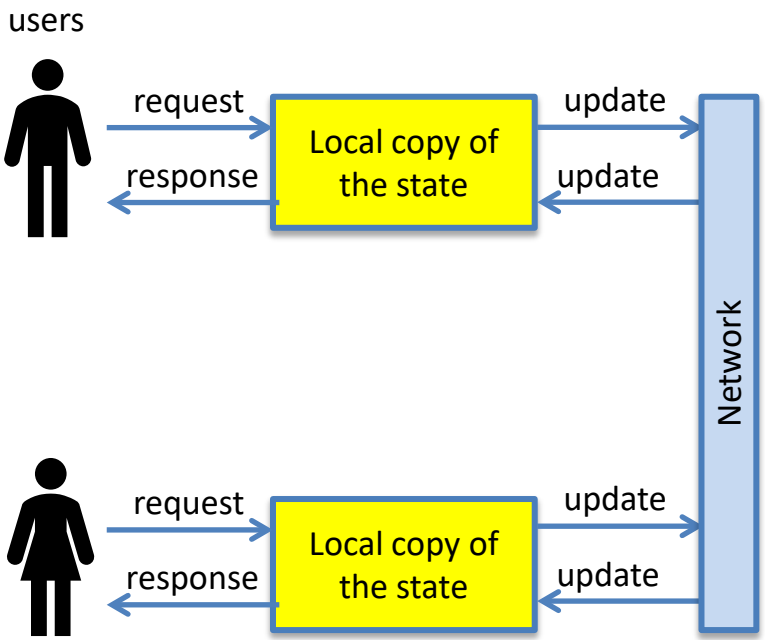
Problems

- Single point of failure
- Unfair (network latency)
- High latency

CAP Theorem: Cannot have

- Consistency
- Availability
- Partition tolerance

Decentralized Solution



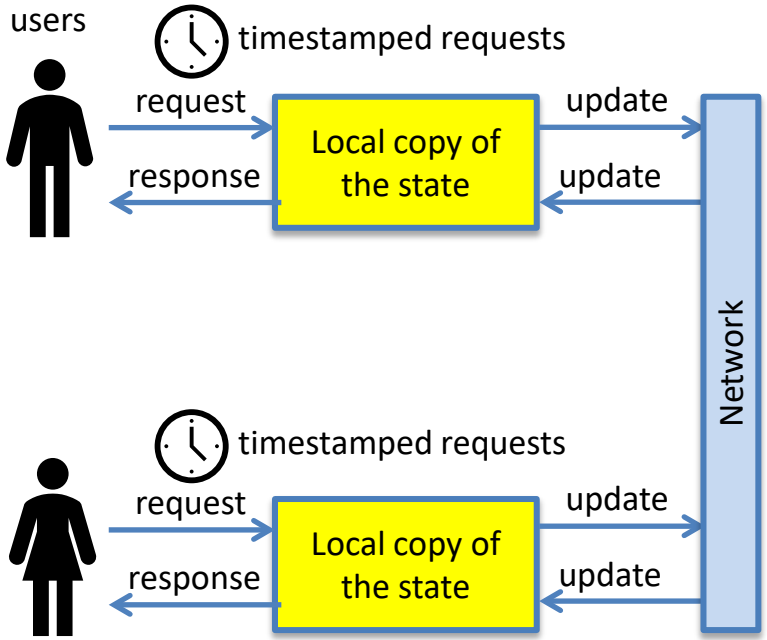
Naïve approaches

- Publish-and-subscribe
- Actors
- Remote procedure calls

Problems

- Inconsistency
- Nondeterminism

Principled Decentralized Solution



Goal:

- Synchronized clocks with bounded error *E*
- Handle events in timestamp order
- Consistency: agreement *at a timestamp*
- Availability: quick responses

Fundamental limit: CAL Theorem:
As ***apparent latency*** increases, either inconsistency or unavailability must increase.
Apparent latency = L + E + X

The Reactor Model of Computation

Reactors:
- Concurrent
- Deterministic
- Distributed
- Time sensitive

https://reactor-model.org

Coordination languages

Programming APIs

https://lf-lang.org

https://lf-lang.org/reactor-uc

https://xronos.com

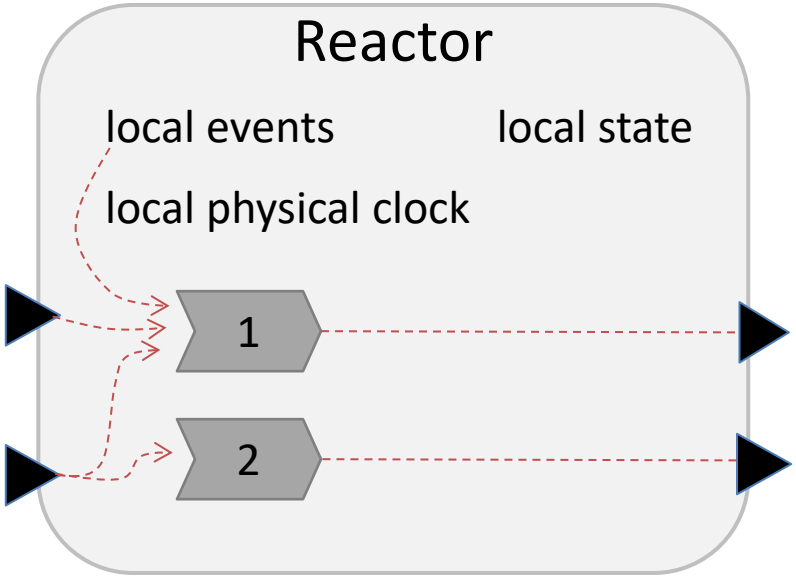
https://rosia.dev

The Reactor Model for Distributed Programs

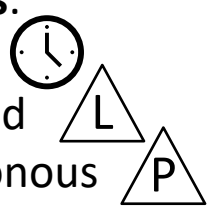
Input ports:
Timestamped events are received in timestamp order at each port.

Reactions:
Triggered by local and/or input events. Manipulate state.

Output ports:
Written to by reactions. Timestamp matches the trigger.



Local events:
- Periodic
- Scheduled
- Asynchronous



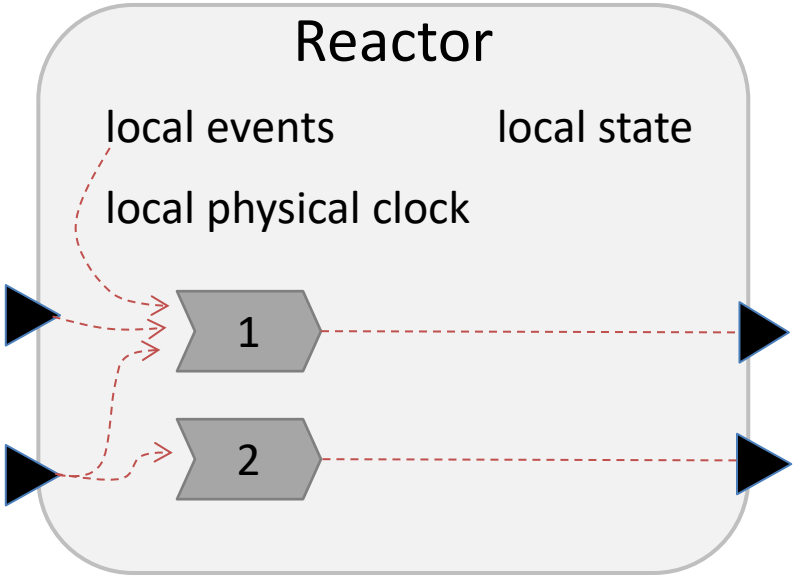
Local physical clock:
Local events are timestamped using this clock.

Goal is to process events in timestamp order.
When events are simultaneous, use reaction order.

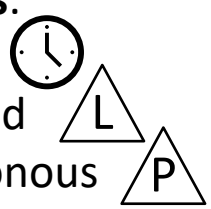
Execution

Input ports:
Timestamped events are received in timestamp order at each port.

Reactions:
Triggered by local and/or input events. Manipulate state.



Local events:
- Periodic
- Scheduled
- Asynchronous



Local physical clock:
Local events are timestamped using this clock.

Output ports:
Written to by reactions.
Timestamp matches the trigger.

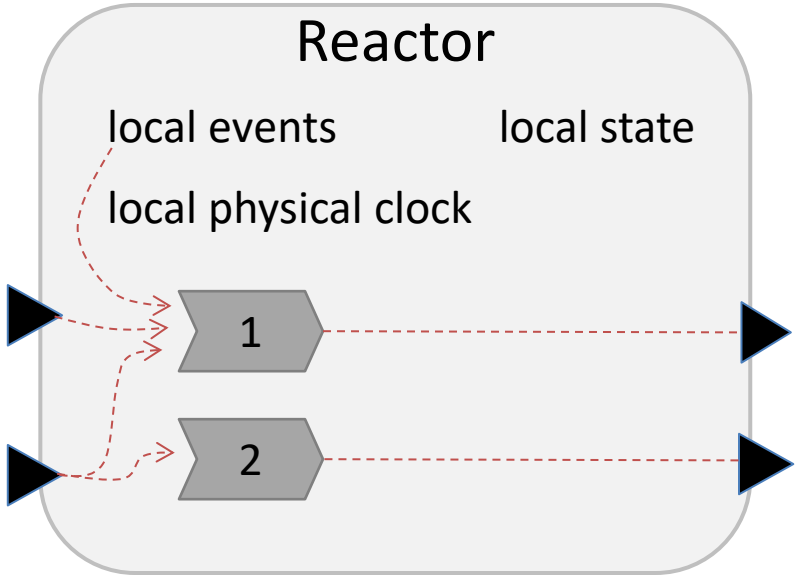
Advance local logical time to the timestamp *t* of the earliest next event. Execute triggered reactions in order.

Logical Time Chases Physical Time

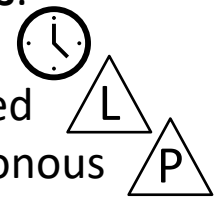
Input ports:
Timestamped events are received in timestamp order at each port.

Reactions:
Triggered by local and/or input events. Manipulate state.

Output ports:
Written to by reactions. Timestamp matches the trigger.



Local events:
- Periodic
- Scheduled
- Asynchronous



Local physical clock:
Local events are timestamped using this clock.

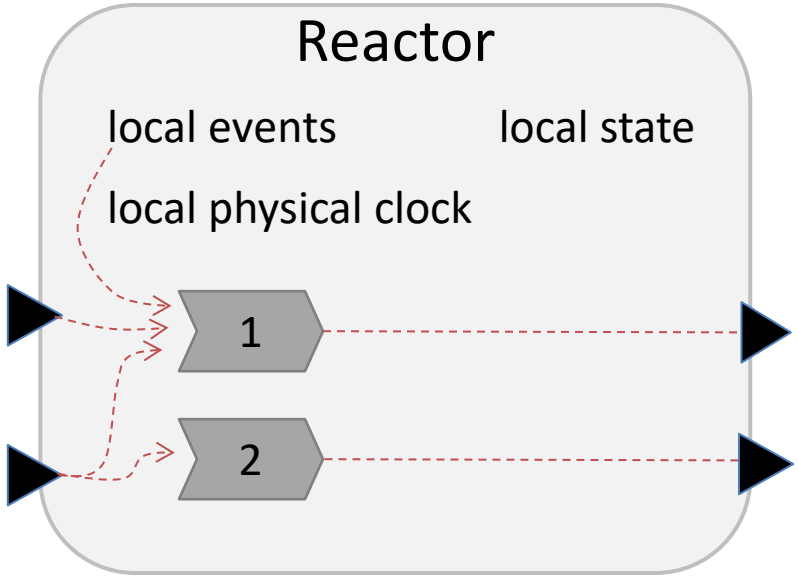
If local events are timestamped using the local physical clock, advancing to logical time *t* cannot occur before local physical time exceeds *t*.

In-Order Network Delivery

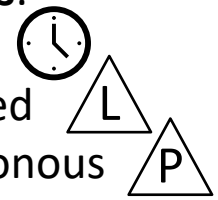
Input ports:
Timestamped events are received in timestamp order at each port.

Reactions:
Triggered by local and/or input events. Manipulate state.

Output ports:
Written to by reactions. Timestamp matches the trigger.



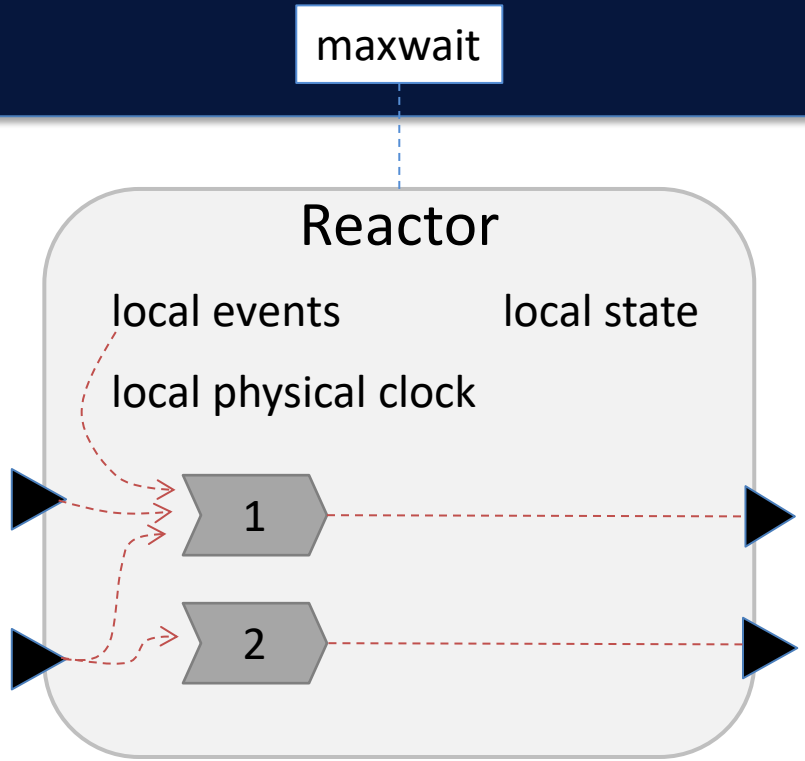
Local events:
- Periodic
- Scheduled
- Asynchronous



Local physical clock:
Local events are timestamped using this clock.

It is safe to advance to logical time *t* when local physical time exceeds *t* and inputs have been received on each input port with timestamp *t* or greater.

maxwait

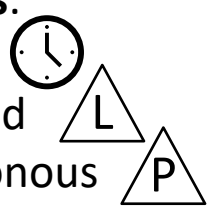


Input ports:
Timestamped events are received in timestamp order at each port.

Reactions:
Triggered by local and/or input events. Manipulate state.

Output ports:
Written to by reactions. Timestamp matches the trigger.

Local events:
- Periodic
- Scheduled
- Asynchronous

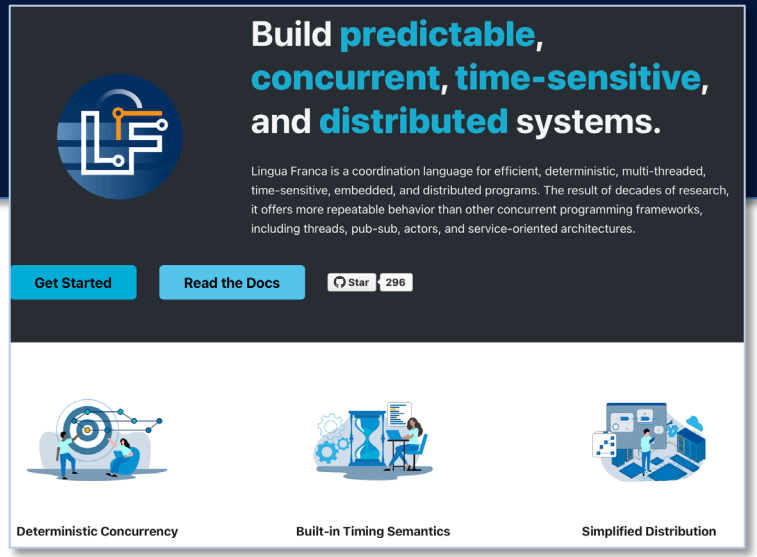


Local physical clock:
Local events are timestamped using this clock.

Advance to logical time *t* when either:
1. Inputs are known up to *t* and *T* > *t* (*T* = local physical time)
2. *T* > *t* + maxwait

Lingua Franca

Concurrent, distributed, timed, & deterministic by default

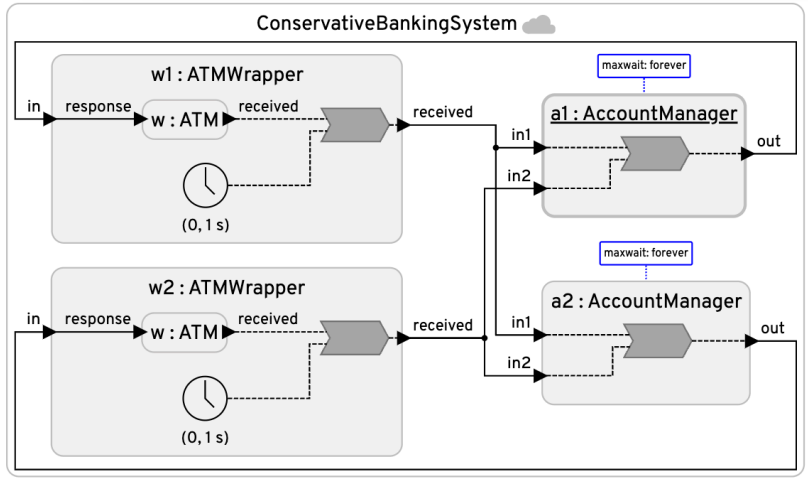


- A polyglot coordination language for high performance, secure, distributed, real-time, safety-critical software.
- An open source tool chain supporting development in C, C++, Python, Rust, and TypeScript: https://github.com/icyphy/lingua-franca
- Shows that time-stamped deterministic discrete-event models can execute efficiently.

- *Deterministic concurrency*
- Automatic multicore utilization
- Federated, distributed execution
- Real-time scheduling

https://lf-lang.org

Visualization of architecture is automatically generated from source.

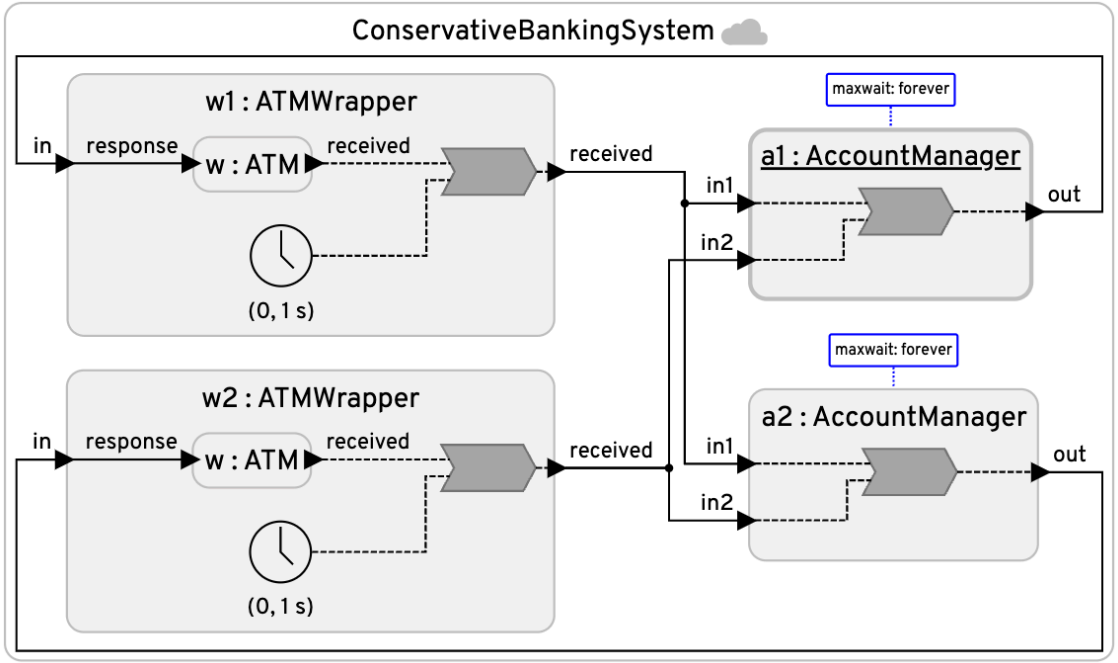


Strong Consistency: maxwait == forever

```
federated reactor {
  w1 = new ATMWrapper()
  w2 = new ATMWrapper()
  @maxwait(forever)
  a1 = new AccountManager()
  @maxwait(forever)
  a2 = new AccountManager()

  w1.received -> a1.in1
  w2.received -> a2.in2
  w1.received -> a2.in1
  w2.received -> a1.in2

  a1.out -> w1.in
  a2.out -> w2.in
}
```

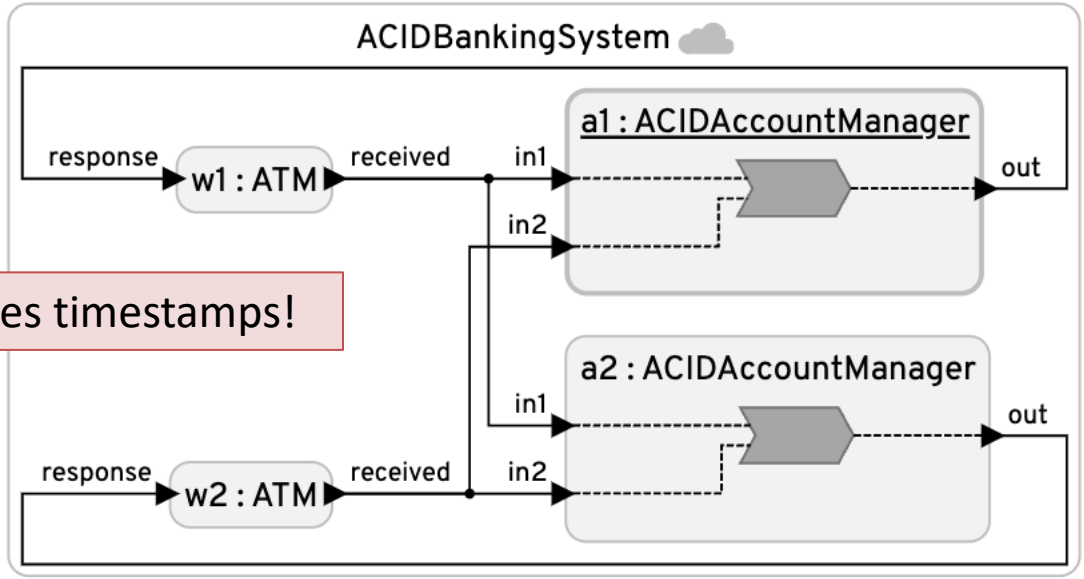


Chandy and Misra [1988] with NULL messages.

Alternative: Bounded maxwait to detect missing NULL messages (heartbeat).

Eventual Consistency Without Coordination

```
reactor ACIDAccountManager {
  input in1: int
  input in2: int
  output out: int
  state balance: int = 0
  reaction(in1, in2) -> out {=
    if (in1->is_present)
      self->balance += in1->value;
    if (in2->is_present)
      self->balance += in2->value;
    lf_set(out, self->balance);
  =} tardy
}
```

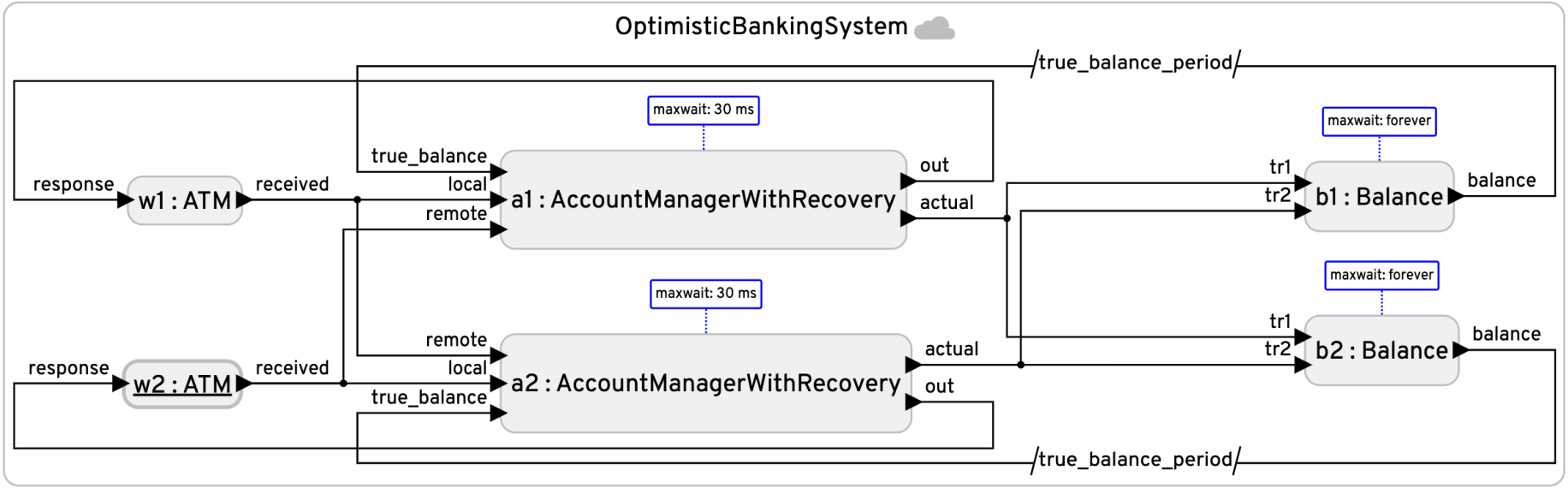


Ignores timestamps!

```
@maxwait(0)
a1 = new ACIDAccountManager()
@maxwait(0)
a2 = new ACIDAccountManager()
```

ACID 2.0 [Helland, 2021], CRDTs [Shapiro, et al., 2011], CALM [Hellerstein & Alvaro, 2021]

Optimistic Execution with Rollback



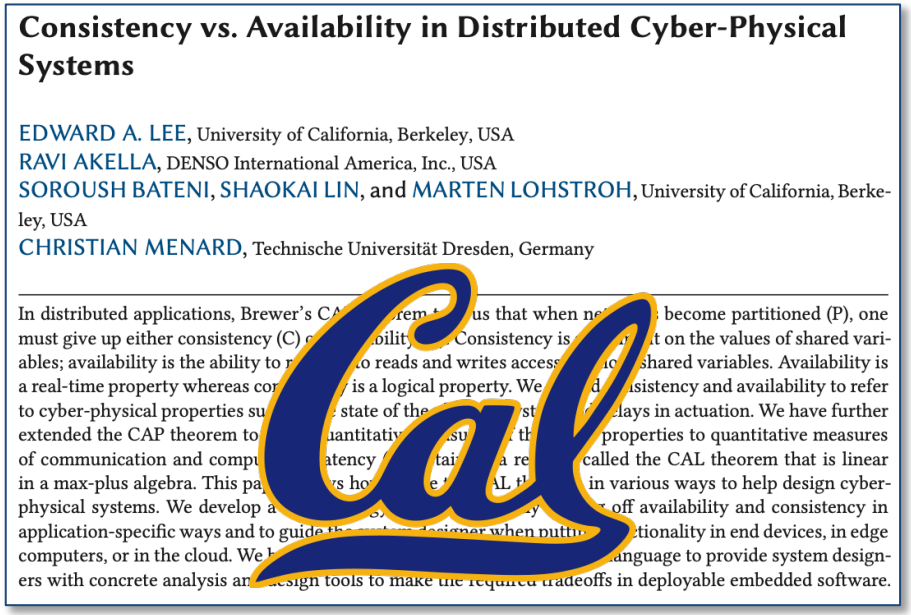
Guarantee 30 ms unavailability, taking measured business risk, and ensure perfect consistency (assuming eventual delivery).

TimeWarp [Jefferson, 1985]

Fundamental Limits

Consistency fundamentally comes with a cost in availability, where the cost is proportional to apparent latency ($L + E + X$).

Reactors with maxwait (and absent_after) enable explicitly managing this tradeoff and detecting faults as early as possible.



Consistency vs. Availability in Distributed Cyber-Physical Systems

EDWARD A. LEE, University of California, Berkeley, USA
RAVI AKELLA, DENSO International America, Inc., USA
SOROUSH BATENI, SHAOKAI LIN, and MARTEN LOHSTROH, University of California, Berkeley, USA
CHRISTIAN MENARD, Technische Universität Dresden, Germany

Lee, E. A., et al. (2023). "Consistency vs. Availability in Distributed Cyber-Physical Systems." ACM Transactions on Embedded Computing Systems (TECS) **22**(5s): 1-24.

Conclusion

All of this depends on clock synchronization with bounded error.

Reactors with maxwait handle a surprising variety of distributed computing patterns:

- Chandy-and-Misra [1988]
- Fault tolerant Chandy-and-Misra
- Logical Execution Time (LET)
- ACID/CRDT/CALM
- Optimistic with or without rollback
- Remote procedure calls (RPC) with futures
- Pub-Sub (maxwait == 0, output port is a topic)
- Actors (maxwait == 0)

Maxwait: A Generalized Mechanism for Distributed Time-Sensitive Systems, by Francesco Paladino, Shulu Li, Edward A. Lee, arXiv:2601.21146, Jan. 2026.