iCyPhy

Software Design for Cyber-Physical Systems

Edward A. Lee

Module 8: Distributed Systems

Technical University of Vienna
Vienna, Austria, May 2022

University of California, Berkeley

Example: Google Spanner
A Globally Distributed Database

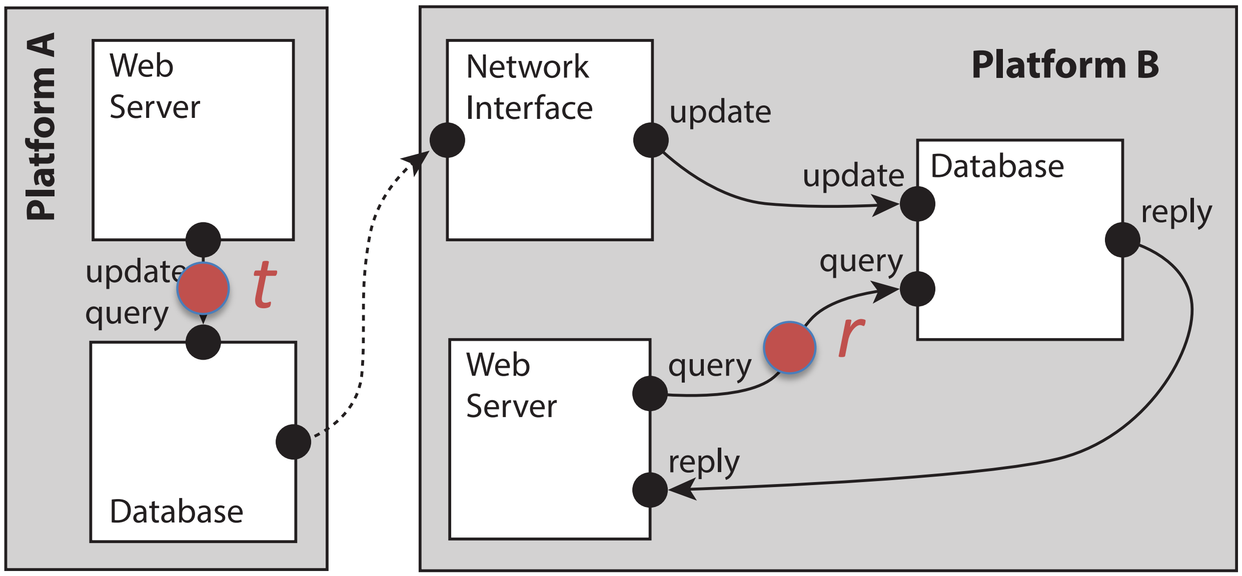
Update to a record comes in. Time stamp *t*.

Distributed database with redundant storage and query handling across data centers.

Query for the same record comes in. Time stamp *r*.

Example: Google Spanner
A Globally Distributed Database

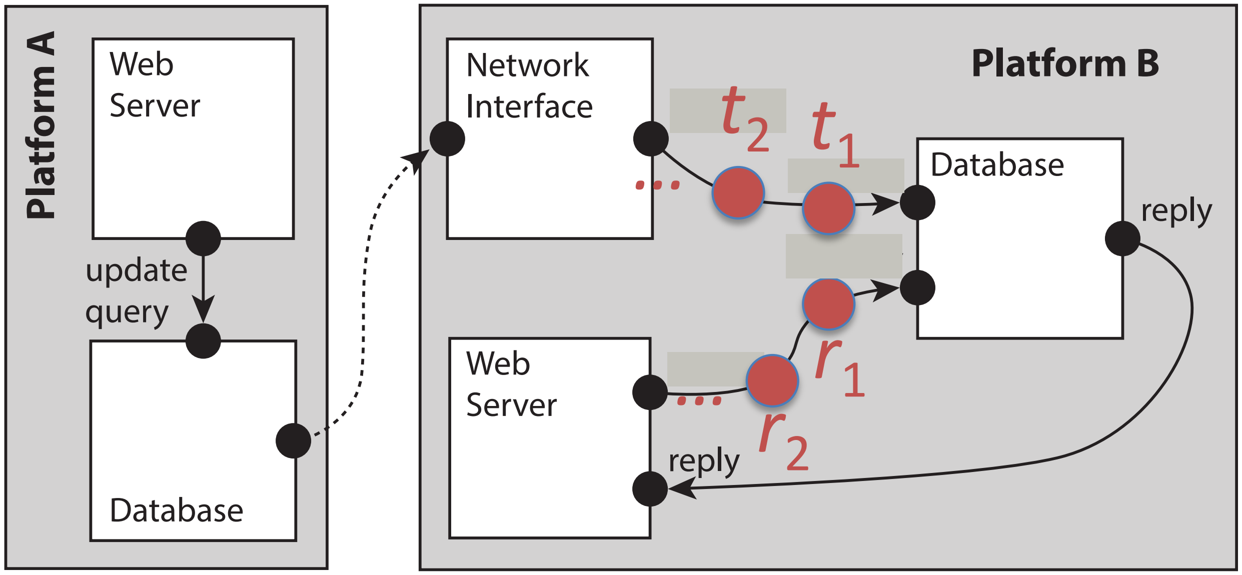
Semantics of the database is that it handles queries in timestamp order.



[Corbet, et al., "Spanner: Google's Globally-Distributed Database," OSDI 2011]

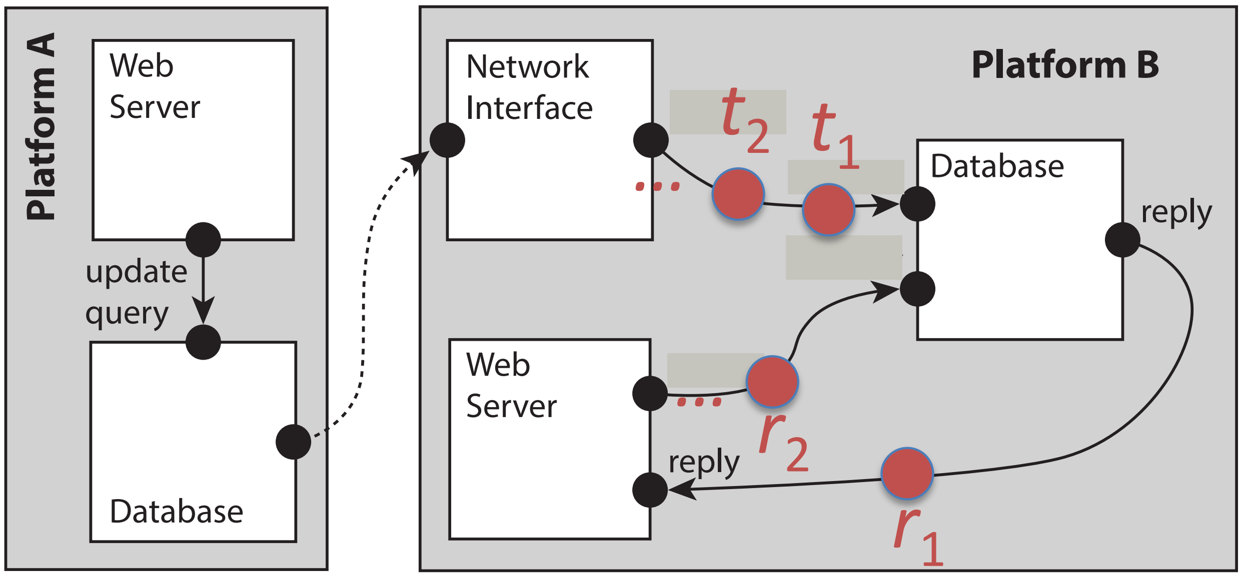
One Possible Approach: Chandy and Misra [1979]

- Assume events arrive reliably in timestamp order.
- Wait for events on each input.
- Process the event with the smaller timestamp.
- E.g. $r_1 < t_1$



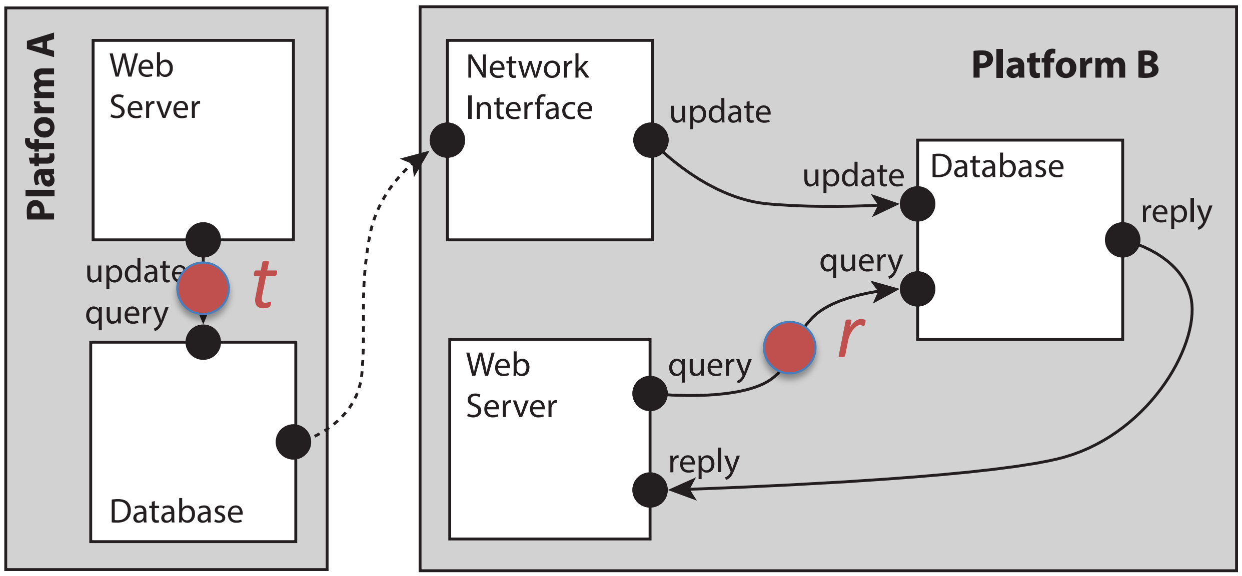
One Possible Approach: Chandy and Misra [1979]

- Deterministic
- Network traffic for "null messages."
- Every node is a single point of failure.



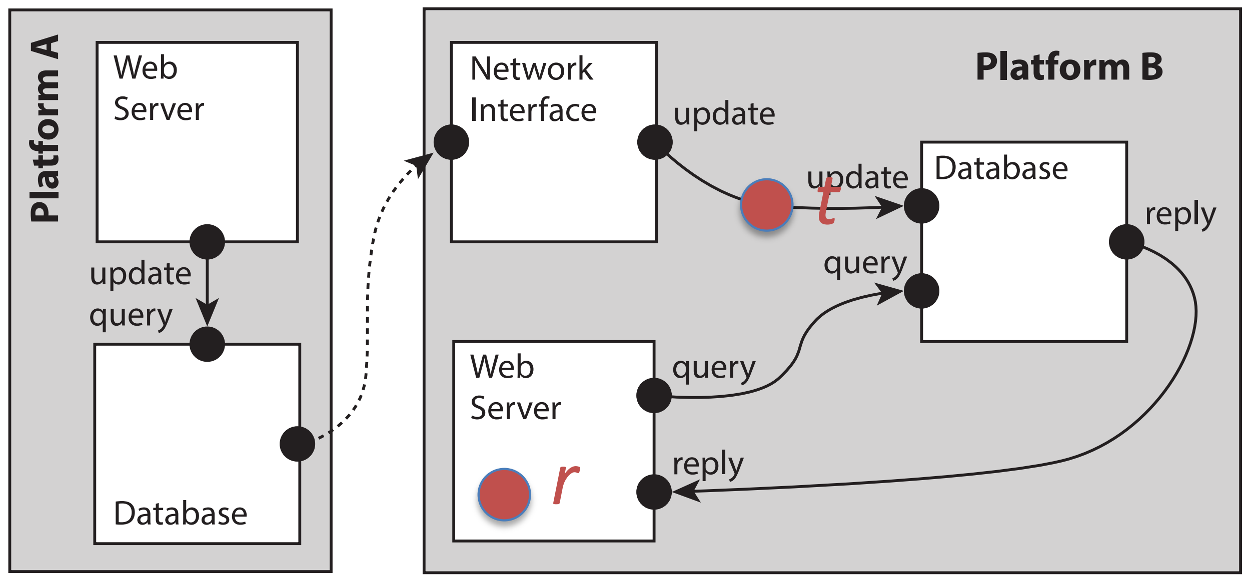
Another Possible Approach: Jefferson: Time Warp [1985]

- Speculatively execute.
- If a message with an earlier timestamp later arrives…



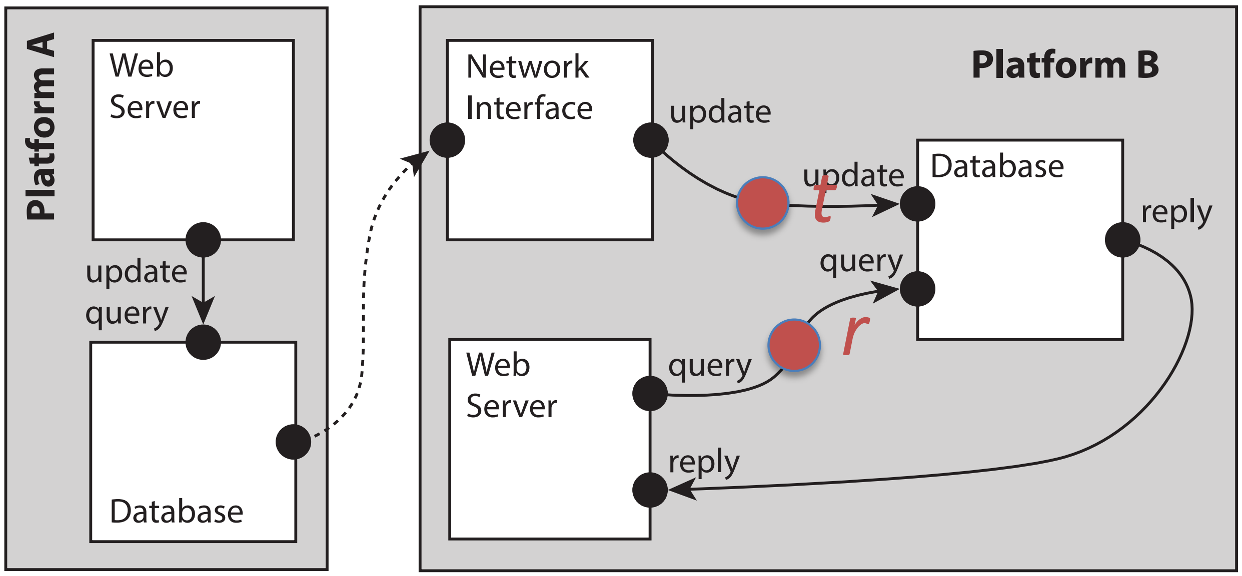
Another Possible Approach: Jefferson: Time Warp [1985]

- Speculatively execute.
- If a message with an earlier timestamp later arrives…
- Backtrack!



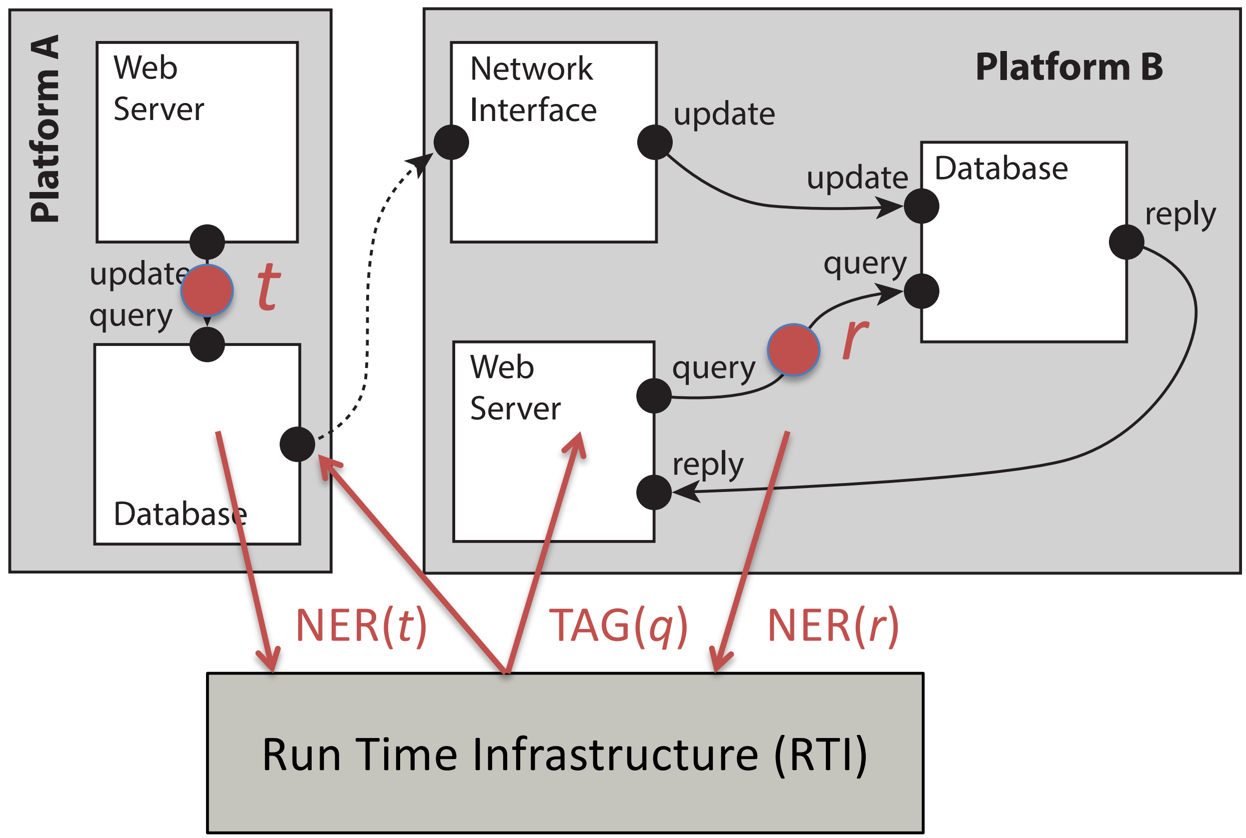
Another Possible Approach: Jefferson: Time Warp [1985]

- No single point of failure.
- Can process events without network traffic
- Can't backtrack side effects.
- Overhead: Snapshots
- Uncontrollable latencies.



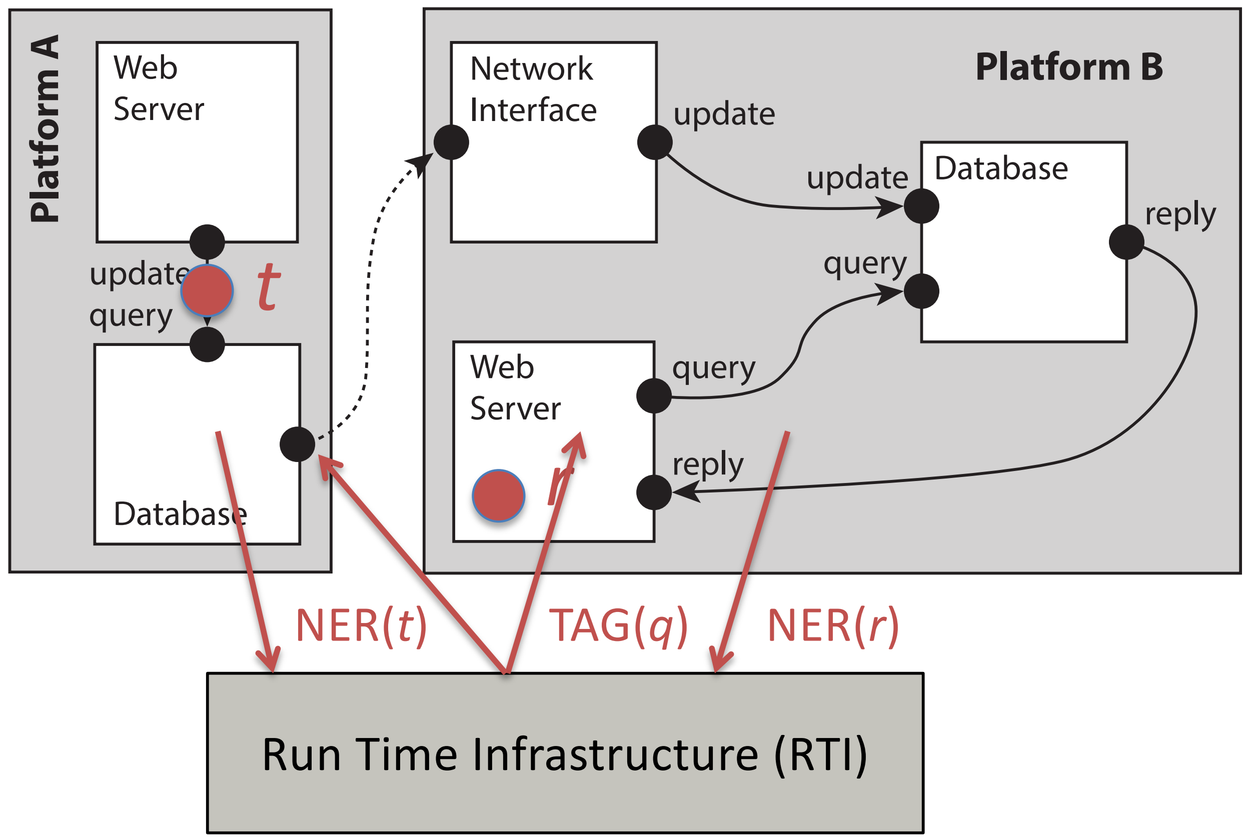
A Third Possible Approach: High Level Architecture (HLA)

- Next event request (NER) with *r*
- Next event request (NER) with *t*
- If $r < t$, then time advance grant (TAG) of $q \leq r$
- If $q = r$, process event



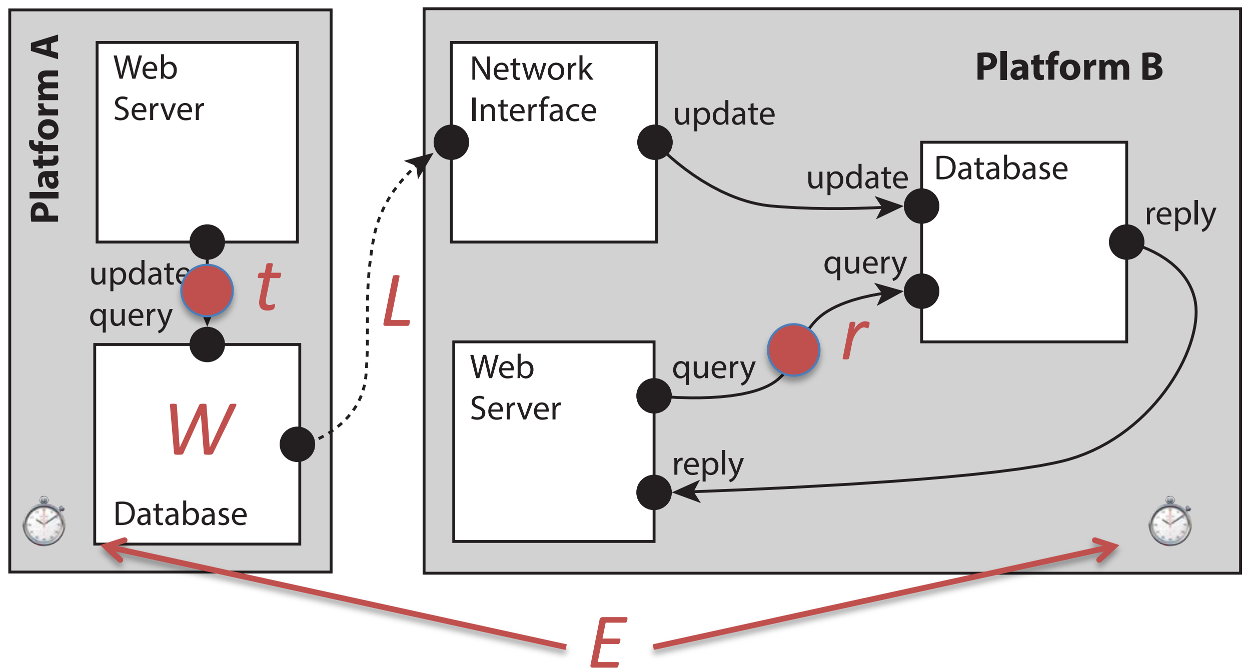
A Third Possible Approach: High Level Architecture (HLA)

- Deterministic.
- RTI is a single point of failure.
- Works well for simulation, but not for online processing.



Ptides/Spanner Approach

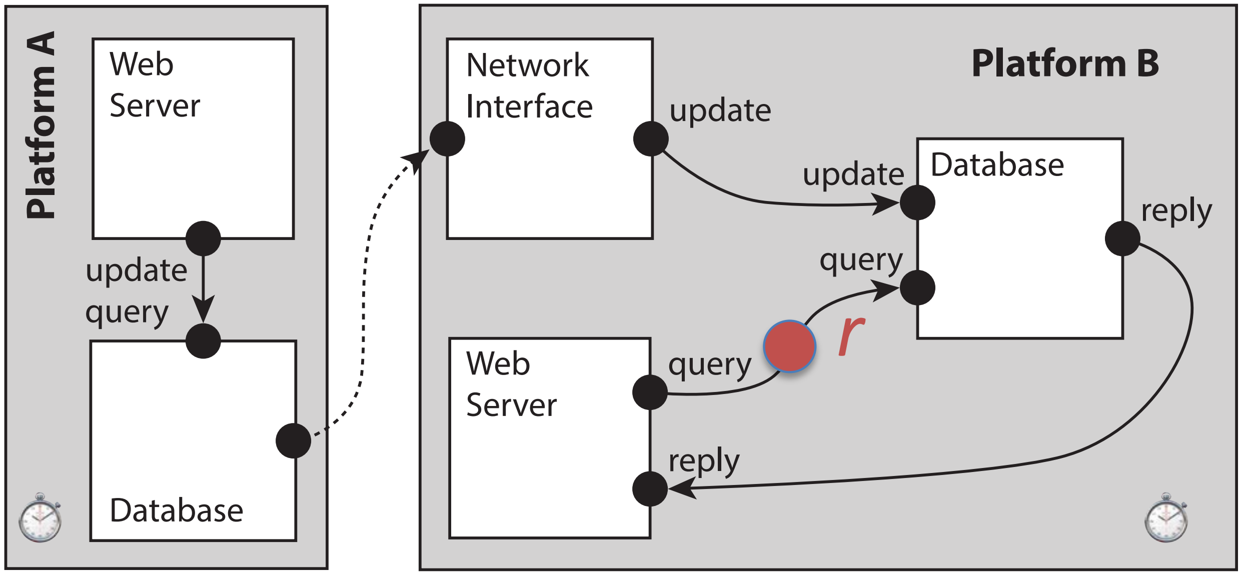
- Local clock on each platform.
- *t* and *r* from local clocks.
- Bounded execution time *W*.
- Bounded network latency *L*.
- Event is known at **B** by time *t* +*W*+*L* (by clock at **A**).
- Bounded clock synchronization error *E*.
- Event is known at **B** by time *t* +*W*+*L*+*E* (by clock at **B**).



Event with timestamp *r* is **safe to process at** time *r* +*W*+*L*+*E* (by clock at **B**).

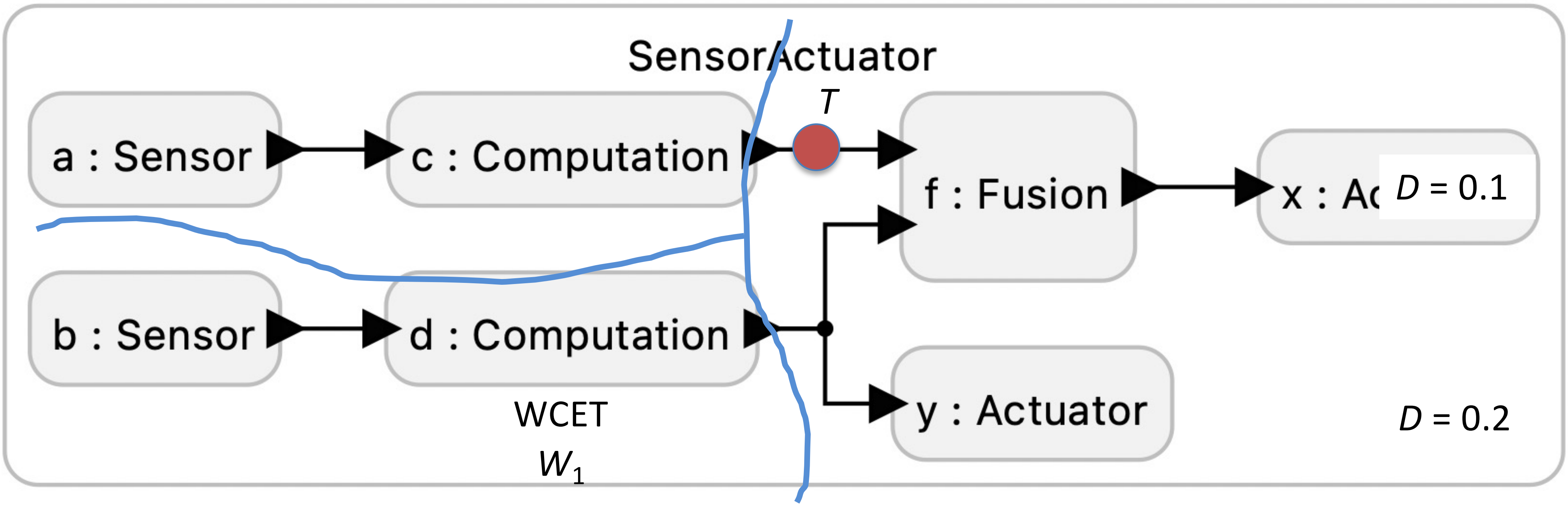
Ptides/Spanner Approach

- No single point of failure.
- Can process events with *no network traffic*.
- Latencies are well defined.
- Time thresholds computed statically.
- Assumptions are clearly stated.



[Zhao, Liu, and Lee, “A Programming Model for Time-Synchronized Distributed Real-Time Systems,” RTAS, 2007]
[Corbet, et al., “Spanner: Google's Globally-Distributed Database,” OSDI 2011]

Networked Scheduling: PTides



When is this "safe to process"?

[Zhao et al., 2007]
[Edison et al., 2012]
[Corbett et al., 2012]

When $T \geq t + W_1 + E + N$, where

- T is the local physical clock time
- W_1 is worst-case execution time
- E is the bound on the clock synchronization error
- N the bound on the network delay

Roots of the Idea

Using Time Instead of Timeout for Fault-Tolerant Distributed Systems

LESLIE LAMPORT
SRI International

A general method is described for implementing a distributed system with any desired degree of fault-tolerance. Instead of relying upon explicit timeouts, processes execute a simple clock-driven algorithm. Reliable clock synchronization and a solution to the Byzantine Generals Problem are assumed.

Categories and Subject Descriptors: C.2.4 [**Computer-Communications Networks**]: Distributed Systems—*network operating systems*; D.1.3 [**Programming Techniques**]: Concurrent Programming; D.4.1 [**Operating Systems**]: Process Management—*synchronization*; D.4.3 [**Operating Systems**]: File Systems Management—*distributed file systems*; D.4.5 [**Operating Systems**]: Reliability—*fault-tolerance*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*; *real-time systems*

General Terms: Design, Reliability

Additional Key Words and Phrases: Clocks, transaction commit, timestamps, interactive consistency, Byzantine Generals Problem

ACM Transactions on Programming Languages and Systems, 1984.

Ptides – A Robust Distributed DE MoC for IoIT Applications

in Proceedings of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 07) , Bellevue, WA, United States.

A Programming Model for Time-Synchronized Distributed Real-Time Systems

Yang Zhao
EECS Department
UC Berkeley

Jie Liu
Microsoft Research
One Microsoft Way

Edward A. Lee
EECS Department
UC Berkeley

Abstract: Discrete-event (DE) models are formal system specifications that have analyzable deterministic behaviors. Using a global, consistent notion of time, DE components communicate via time-stamped events. DE models have primarily been used in performance modeling and simulation, where time stamps are a modeling property bearing no relationship to real time during execution of the model. In this paper, we extend DE models with the capability of relating certain events to physical time…

Google Spanner – A Reinvention

Google independently developed a very similar technique and applied it to distributed databases.

Spanner: Google's Globally-Distributed Database

James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, Dale Woodford

Google, Inc.

Abstract

Spanner is Google's scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. This paper describes how Spanner is structured, its feature set, the rationale underlying various design decisions, and a novel time API that exposes clock uncertainty. This API and its implementation are critical to supporting external consistency and a variety of powerful features: non-blocking reads in the past, lock-free read-only transactions, and atomic schema changes, across all of Spanner.

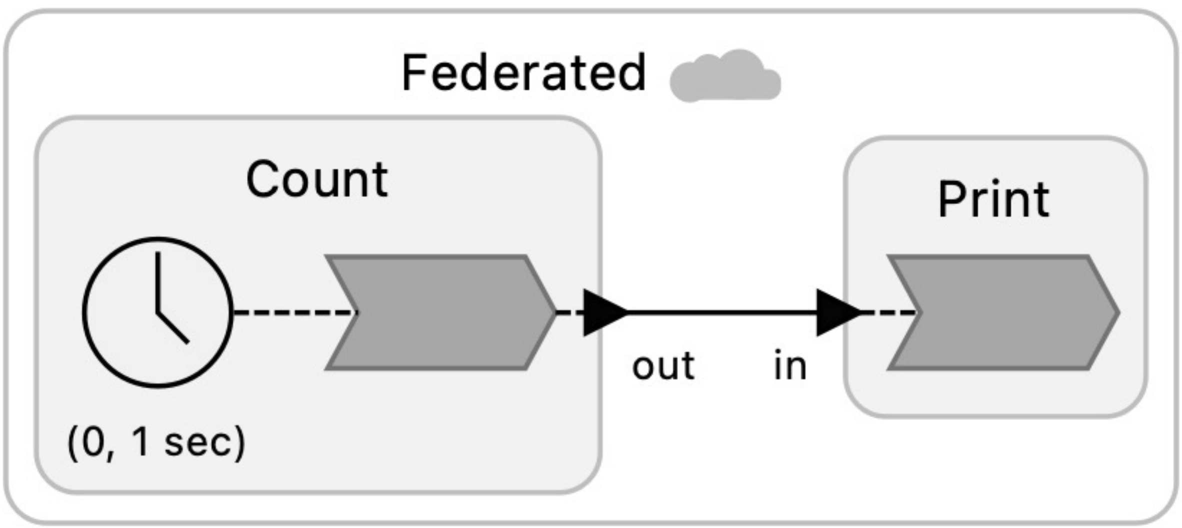
tency over higher availability, as long as they can survive 1 or 2 datacenter failures.

Spanner's main focus is managing cross-datacenter replicated data, but we have also spent a great deal of time in designing and implementing important database features on top of our distributed-systems infrastructure. Even though many projects happily use Bigtable [9], we have also consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications: those that have complex, evolving schemas, or those that want strong consistency in the presence of wide-area replication. (Similar claims have been made by other authors [37].) Many applications at Google

Proceedings of OSDI 2012

Federated LF Programs

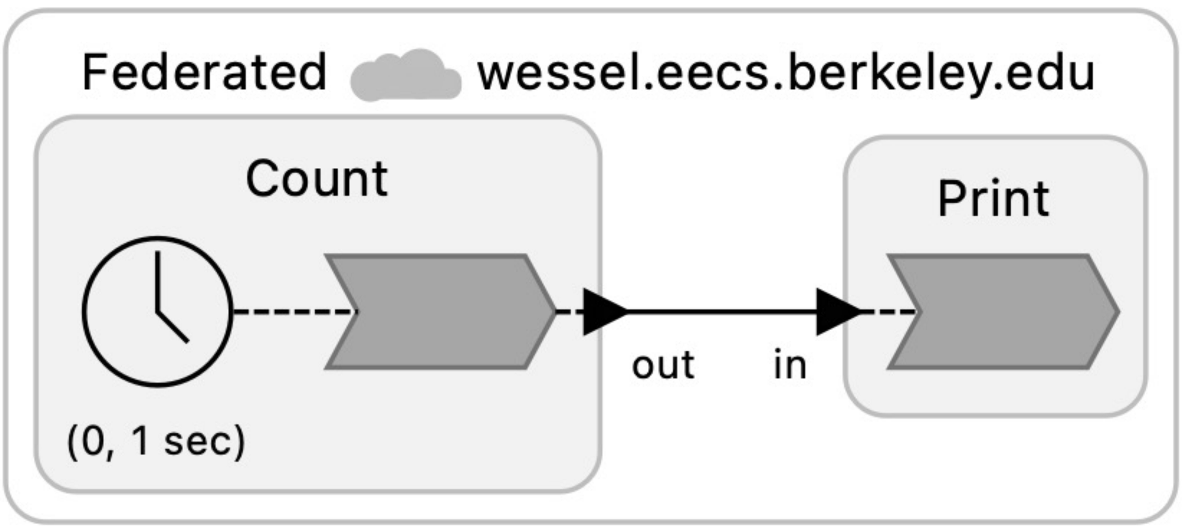
```
federated reactor {
    c = new Count();
    p = new Print();
    c.out -> p.in;
}
```



Federated LF Programs

```
federated reactor at wessel.eecs.berkeley.edu {
    c = new Count();
    p = new Print();
    c.out -> p.in;
}
```

This will put the RTI (runtime infrastructure) on the specified machine. The federates can go anywhere.



Install the RTI

https://lf-lang.org/docs/handbook/distributed-execution

```
git clone https://github.com/lf-lang/reactor-c.git
cd reactor-c/core/federated/RTI/
mkdir build && cd build
cmake ../
make
sudo make install
```

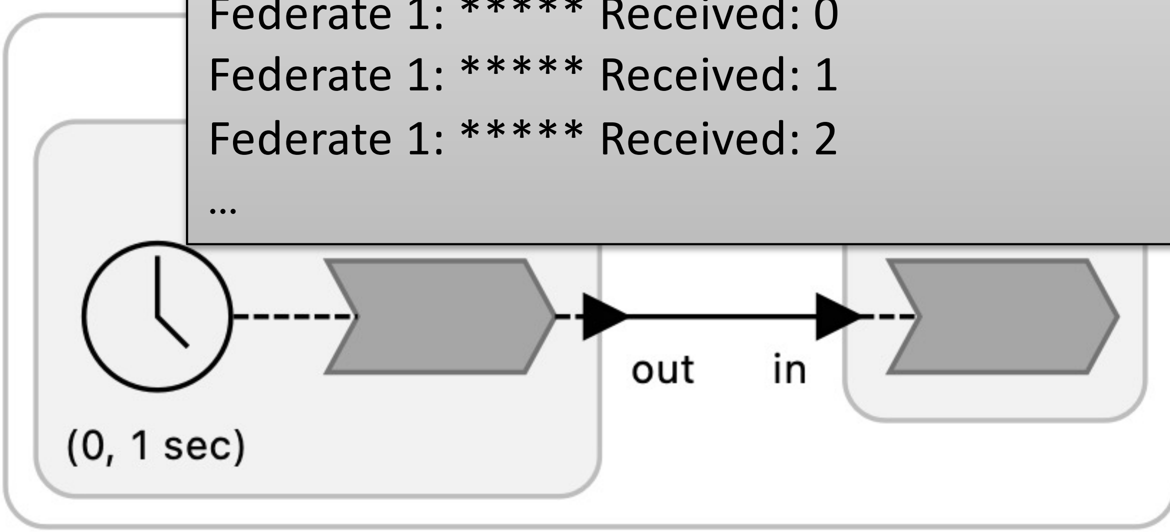
Download Epoch and/or command-line tools from the nightly build (0.2.0 and VS Code extension will not work)

Running by Hand

```
> RTI -n 2 -i myFedID
RTI: Number
RTI: Federatio
Starting RTI f
RTI using TCP
RTI: Listening
…
```

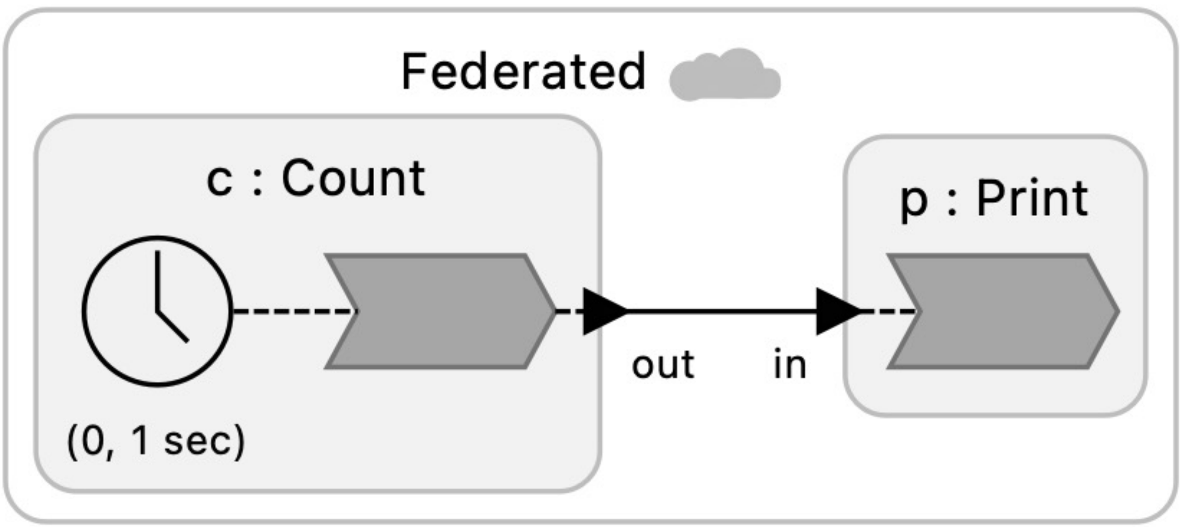
```
> bin/Federated_c -i myFedID
Federation ID
Federate 0: C
---- Start exe
---- plus 4479
Federate 0: -
…
```

```
> bin/Federated_p -i myFedID
Federation ID for executable bin/Federated_p: myFedID
Federate 1: Connected to RTI at localhost:15045.
---- Start execution at time Fri May 13 06:02:19 2022
---- plus 563334000 nanoseconds.
Federate 1: ---- Using 6 workers.
Federate 1: Starting timestamp is: 1652414540563681000.
Federate 1: ***** Received: 0
Federate 1: ***** Received: 1
Federate 1: ***** Received: 2
…
```



Running Using Script

```
> bin/Federated
RTI: Federation ID: 244caf75c3fe2deeda5001d944a256c3637b7c4d796824e5
…
Federate 1: ***** Received: 0
Federate 1: ***** Received: 1
Federate 1: ***** Received: 2
```

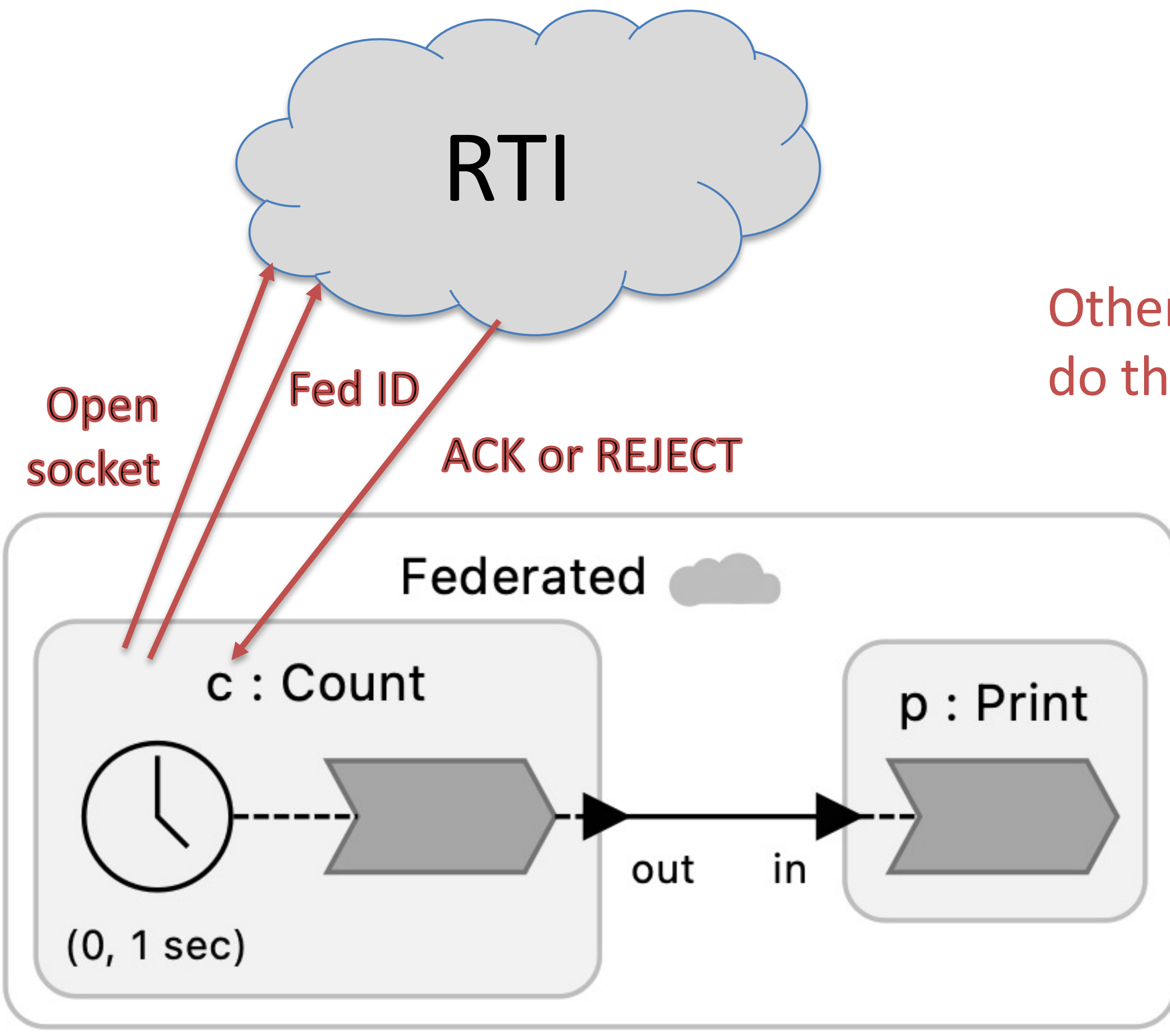


Coordination Across a Distributed System

Centralized: Enforces deterministic semantics regardless of network delays and execution times (based on HLA). (This is the default.)

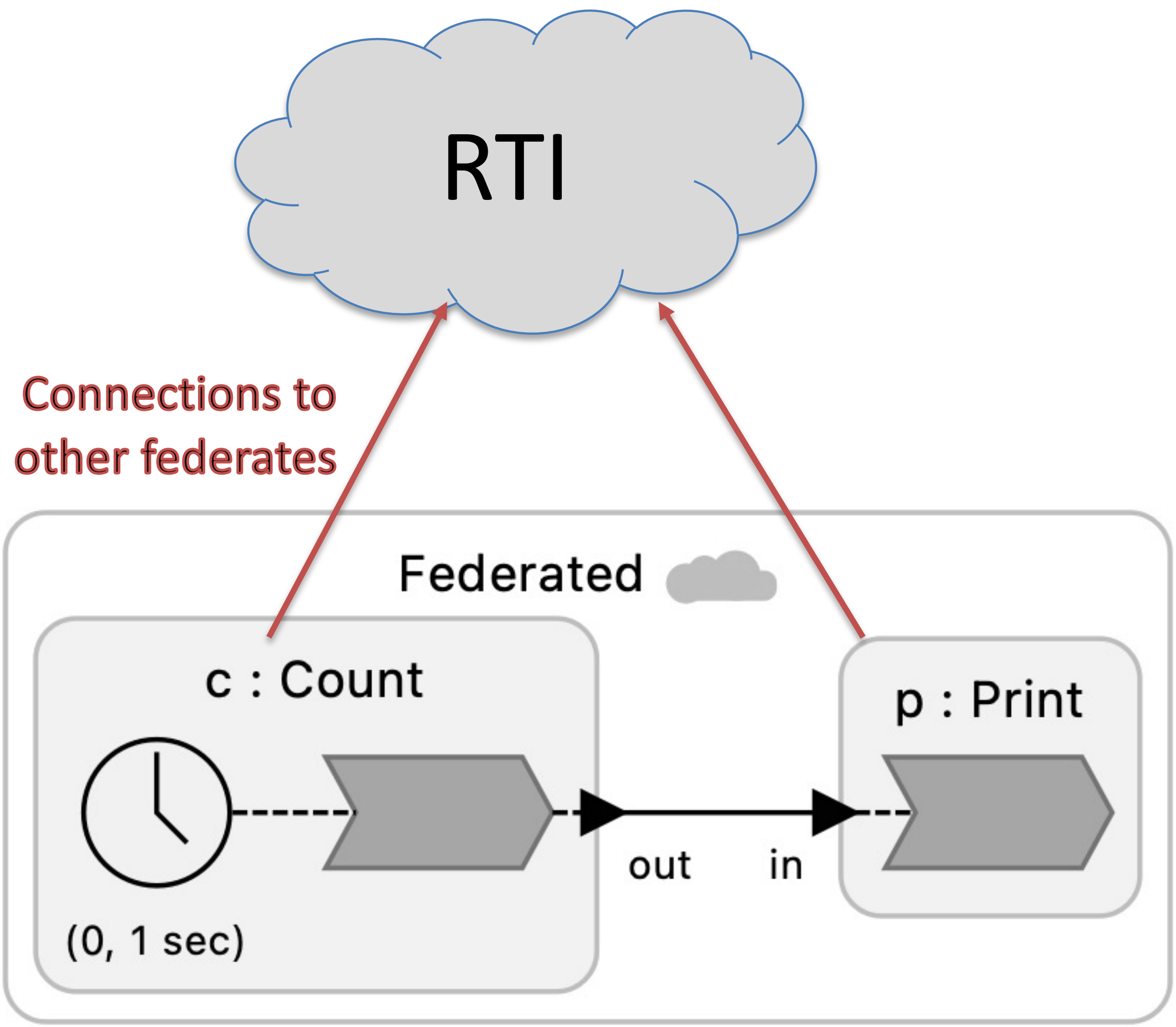
Decentralized: Enforces forward progress and detects violations of deterministic semantics when network delays get too large (based on Ptides).

Federated Startup Initial Connection

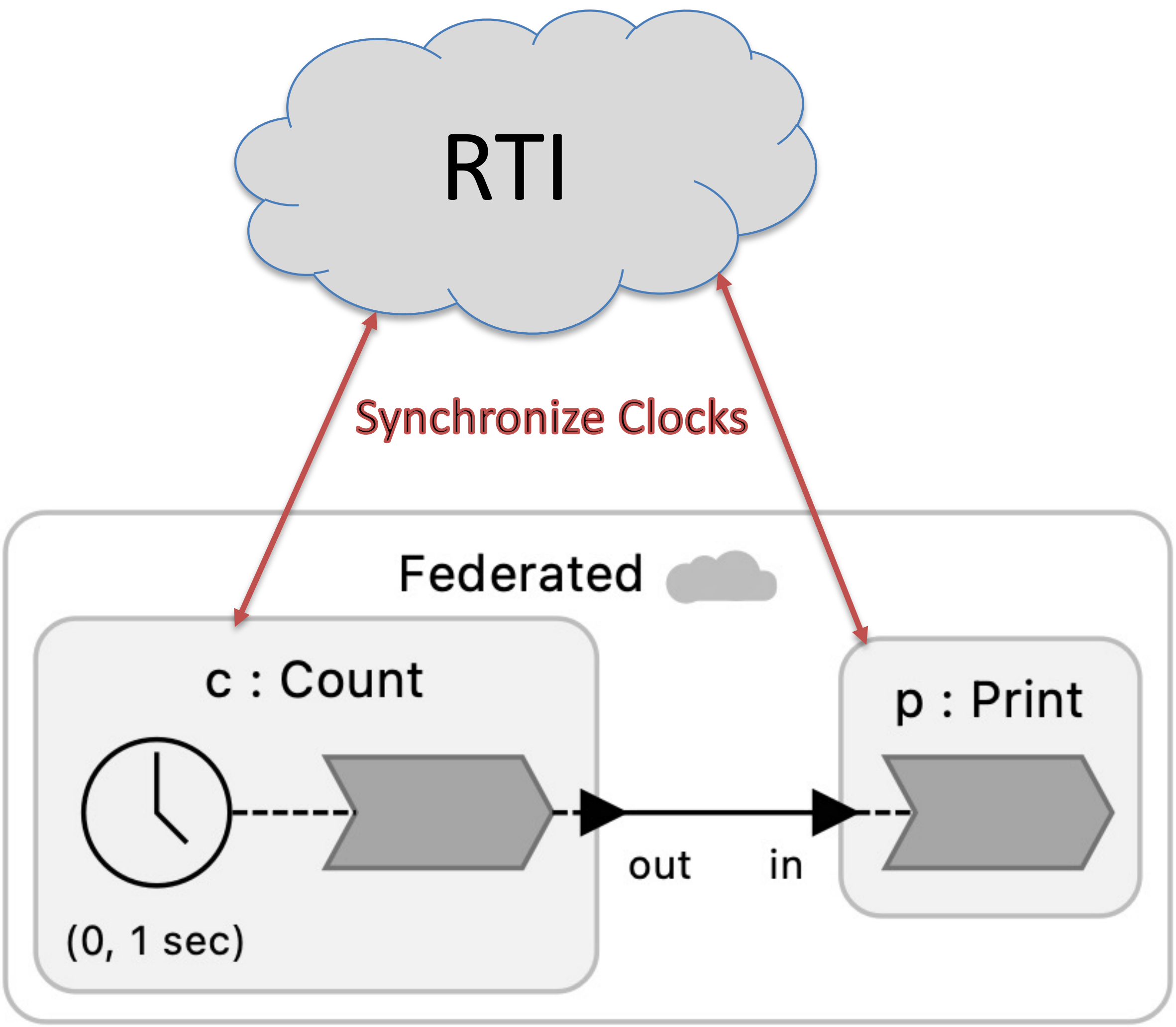


Other federates do the same.

Centralized Coordination Only: Tell RTI of Connection Structure



Clock Synchronization



How Clock Synchronization Works

Precision Time Protocols

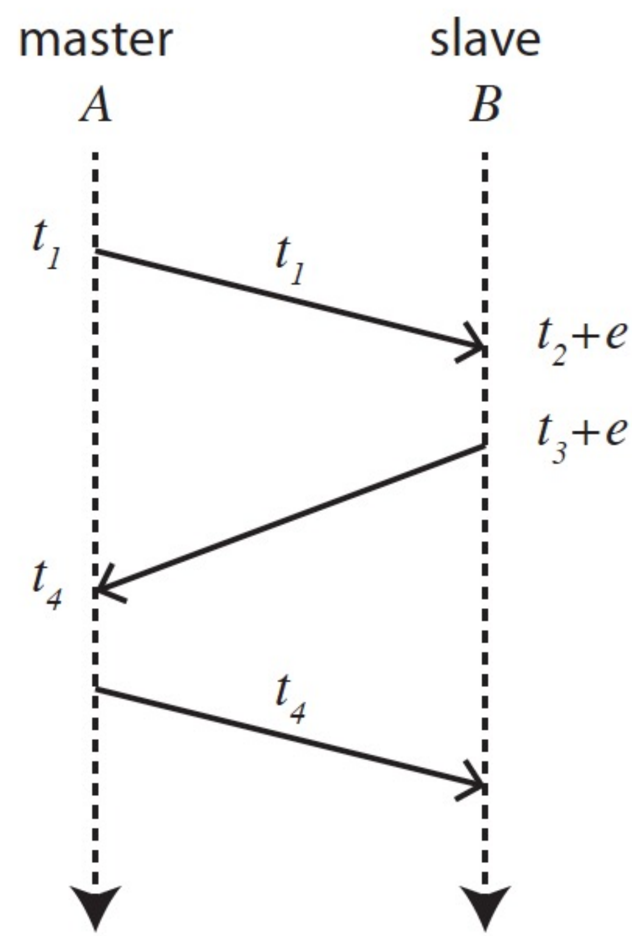
Round-trip delay:

$$r = (t_4 - t_1) - ((t_3 + e) - (t_2 + e)).$$

where e is the clock error in the slave. Estimate of the clock error is

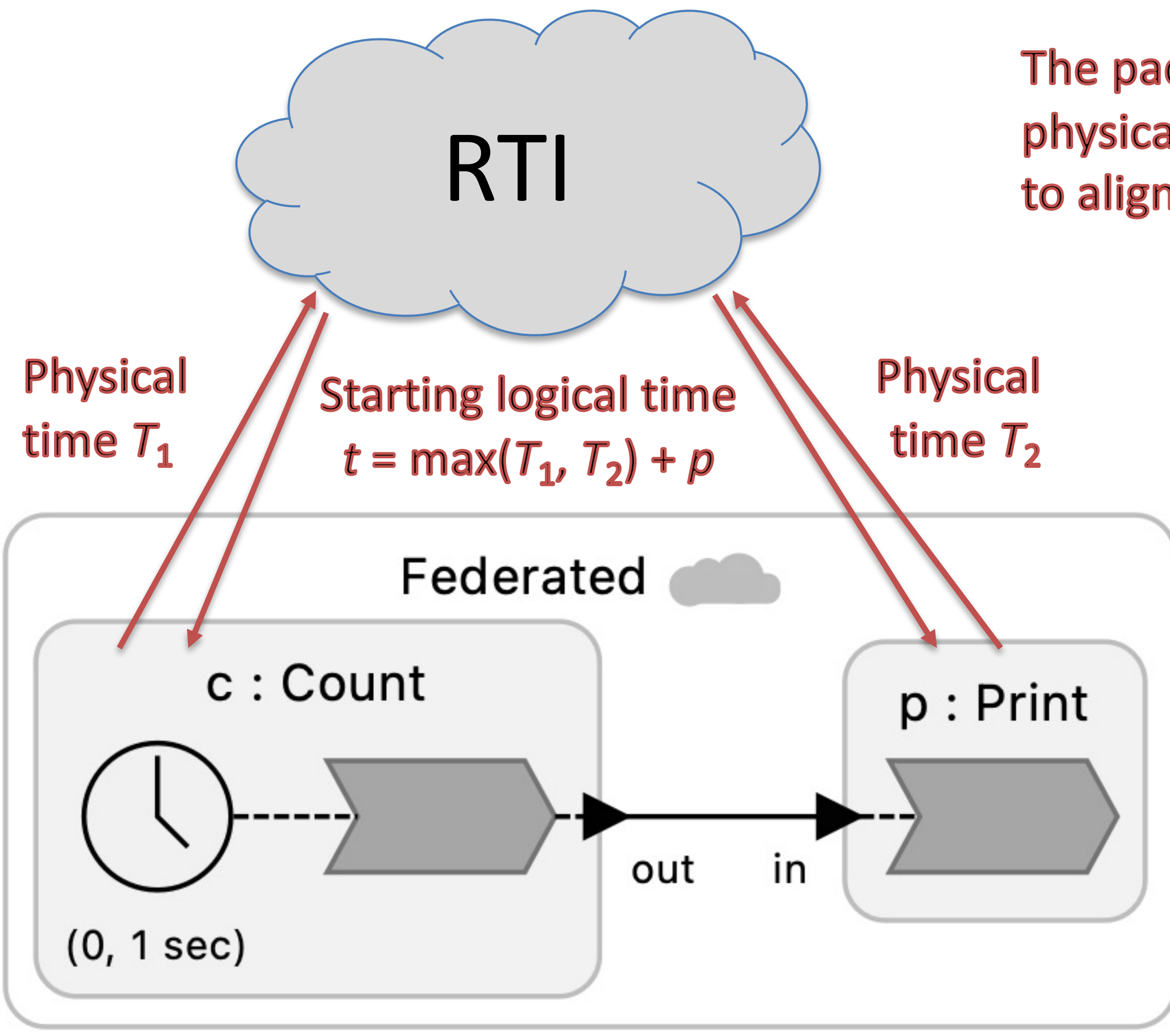
$$\tilde{e} = (t_2 + e) - t_1 - r/2.$$

If communication latency is exactly symmetric, then $\tilde{e} = e$, the exact clock error. B calculates $\tilde{e}$ and adjusts its local clock.



IEEE 1588,
IEEE 802.1AS

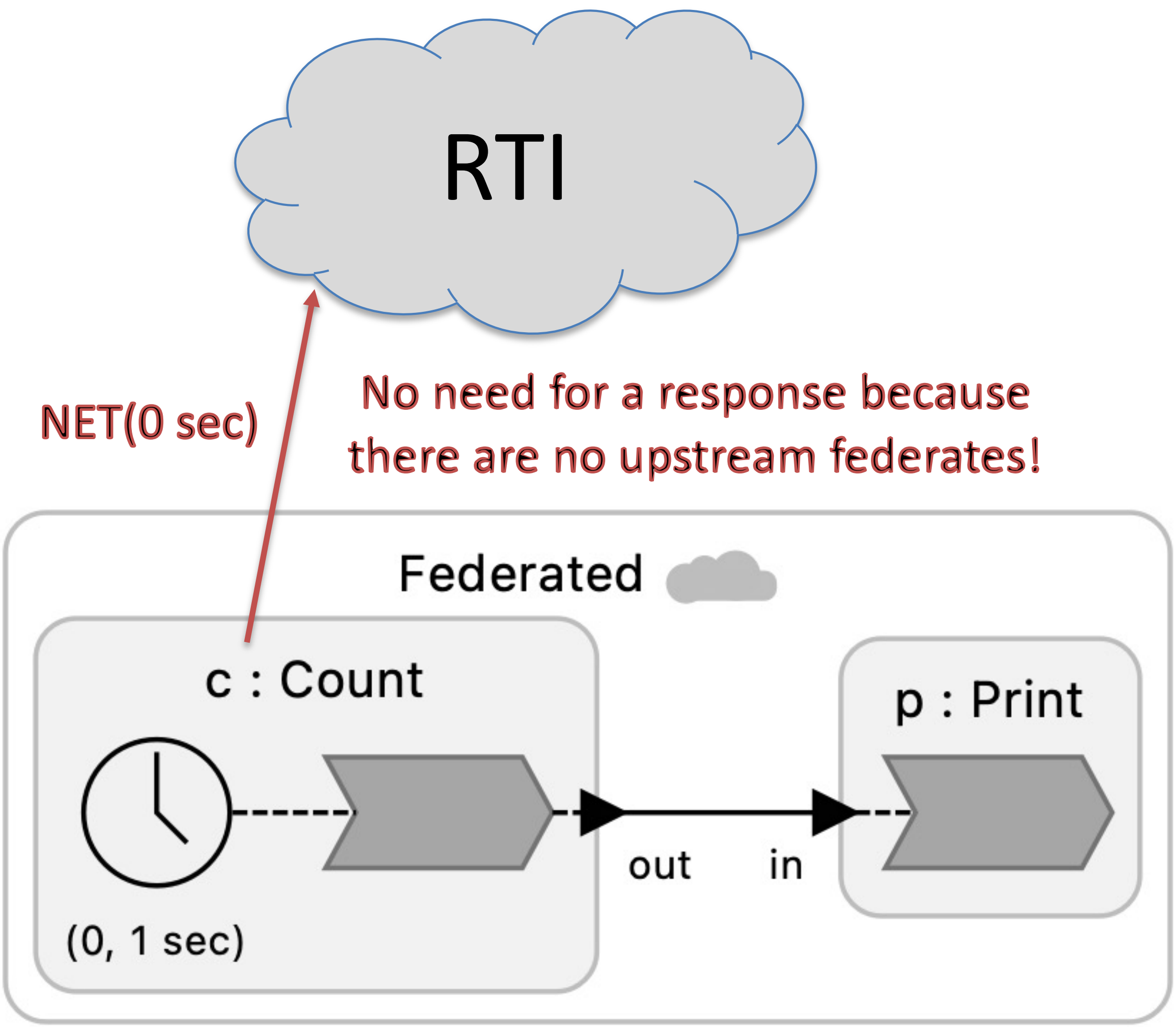
Federated Startup:
Determining the Starting Logical Time



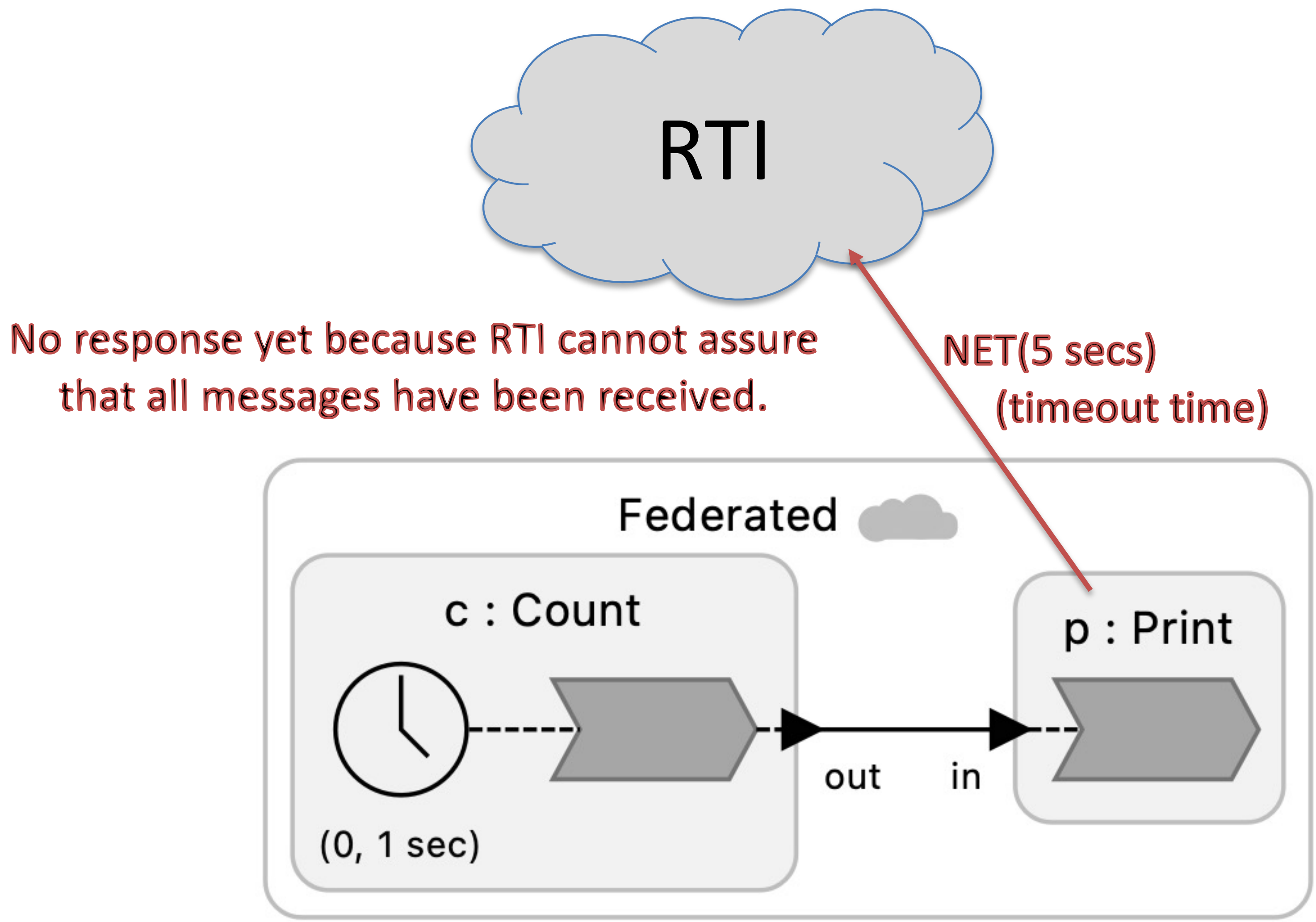
The padding p helps physical and logical times to align well at startup.

Centralized Coordination : Next Event Request (NET)

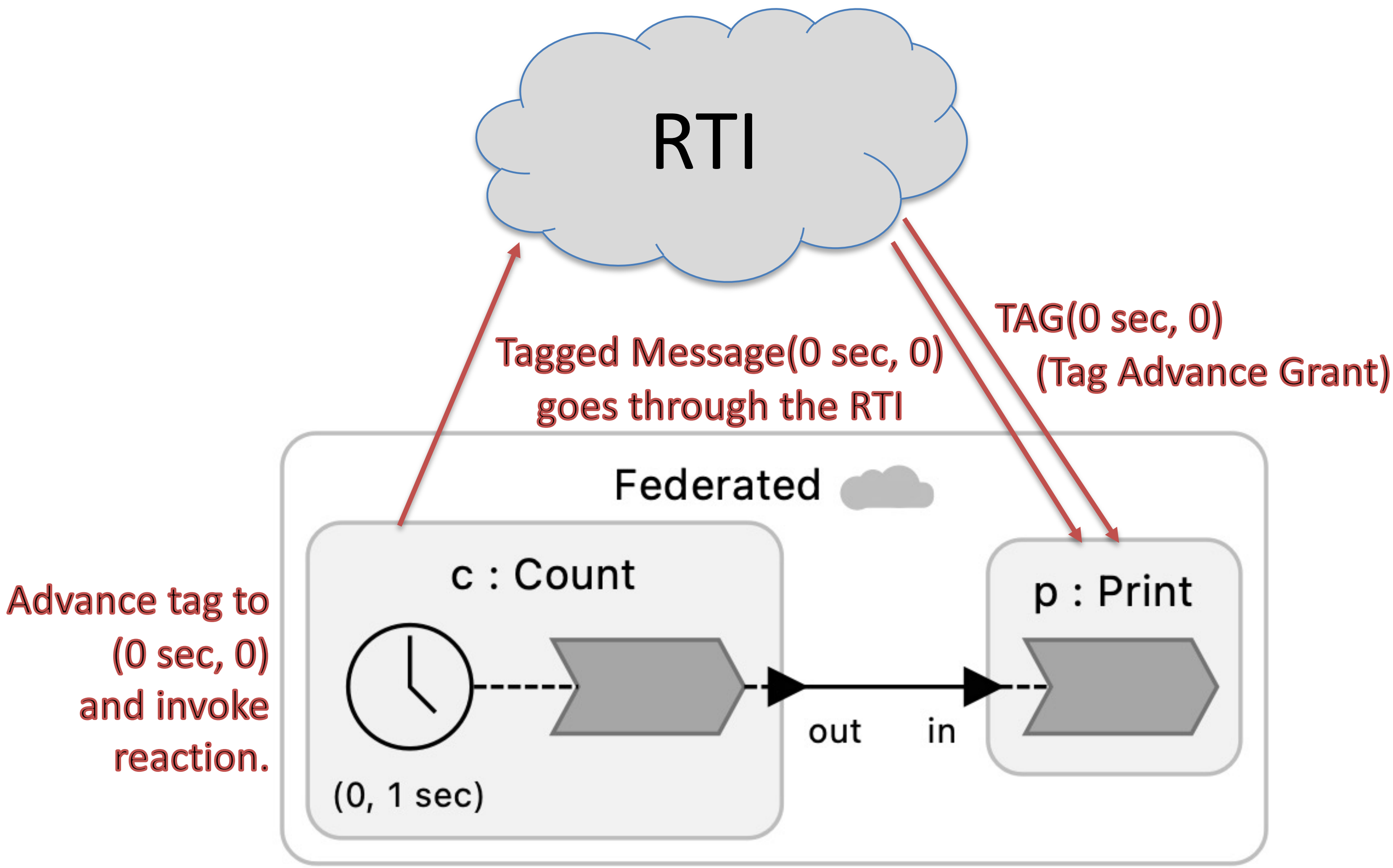
NET: Next Event Tag



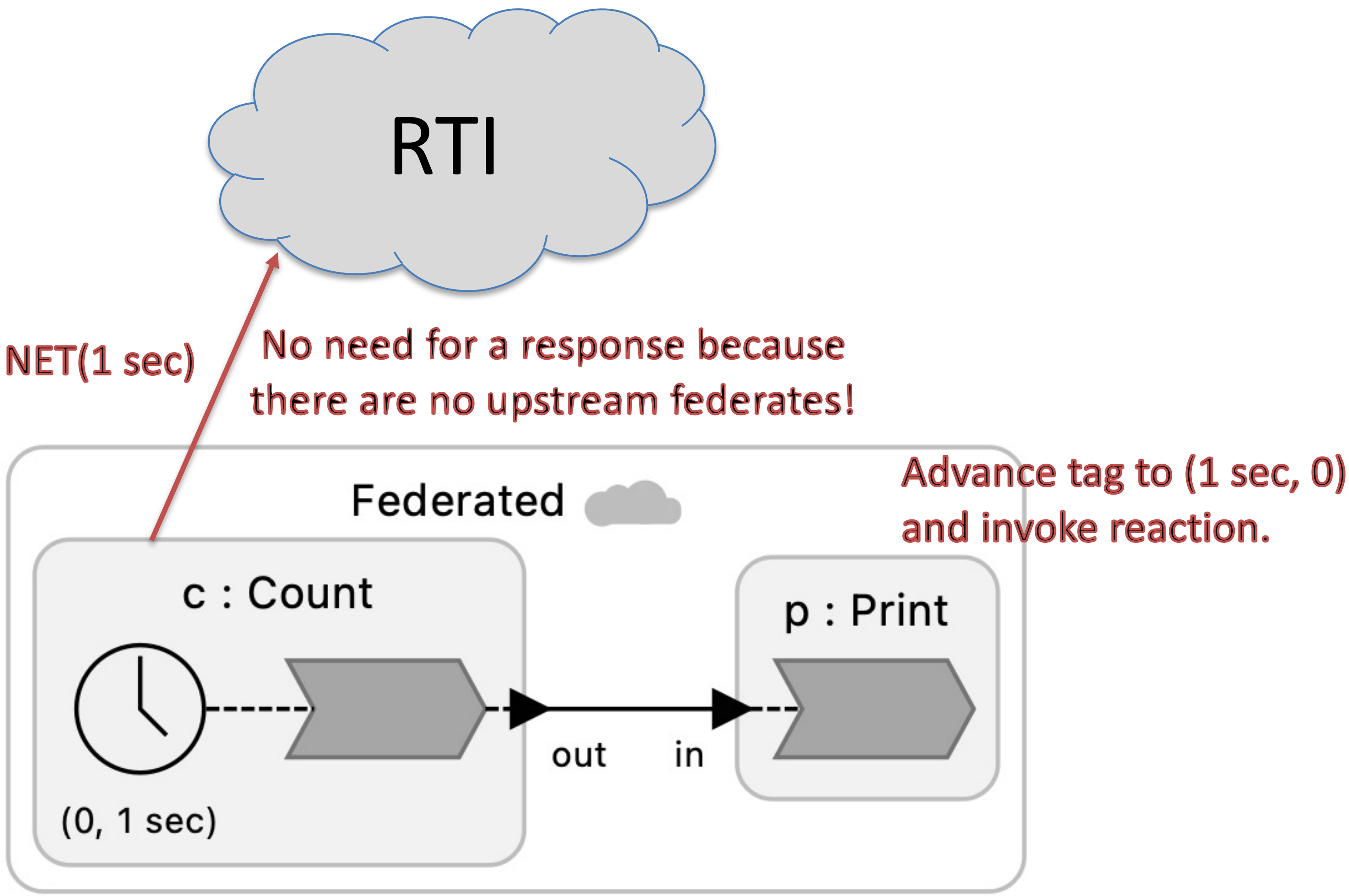
Centralized Coordination:
Next Event Request (NET)



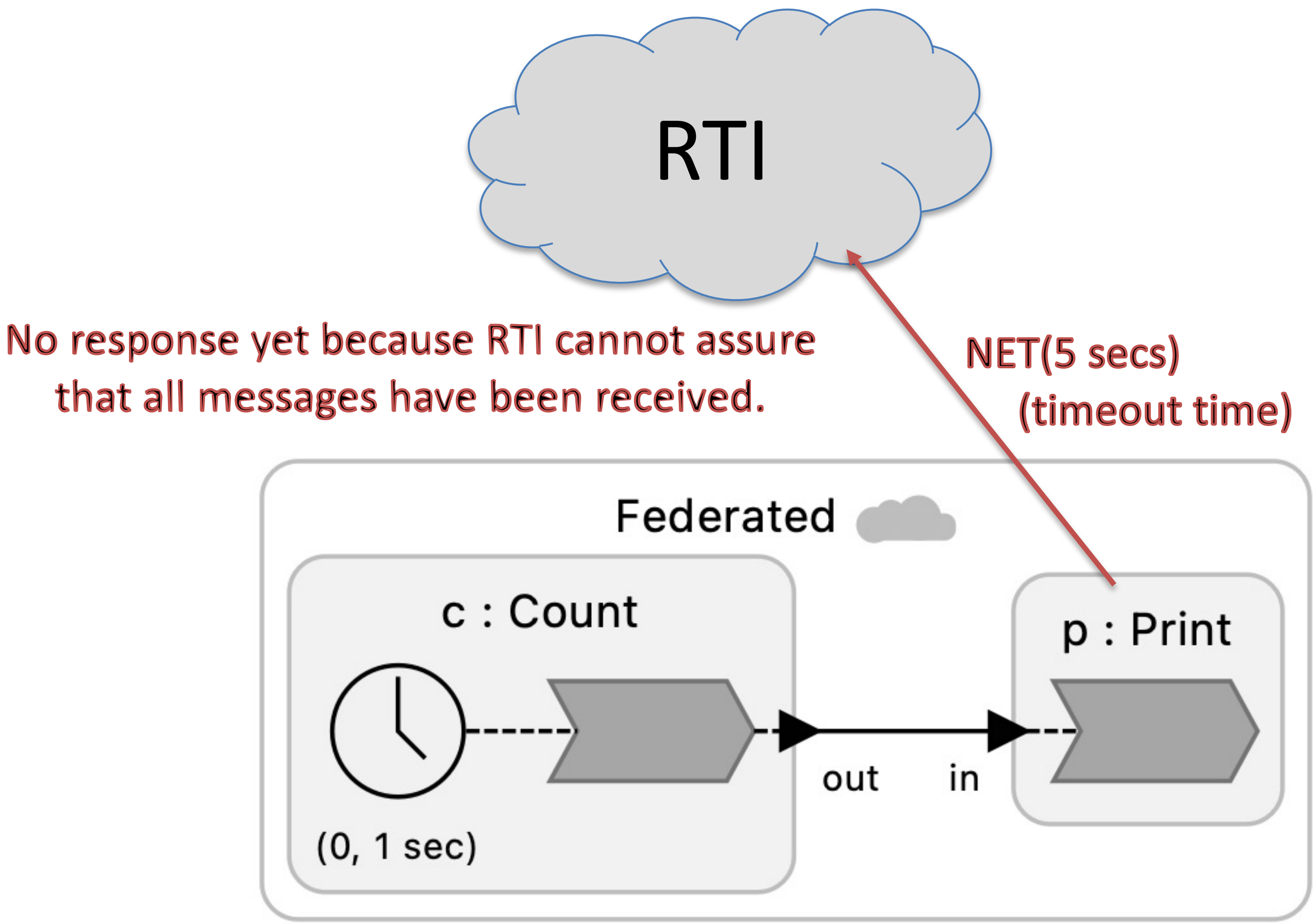
Centralized Coordination : Tagged Message Sending via RTI



Centralized Coordination : Next Event Request (NET)

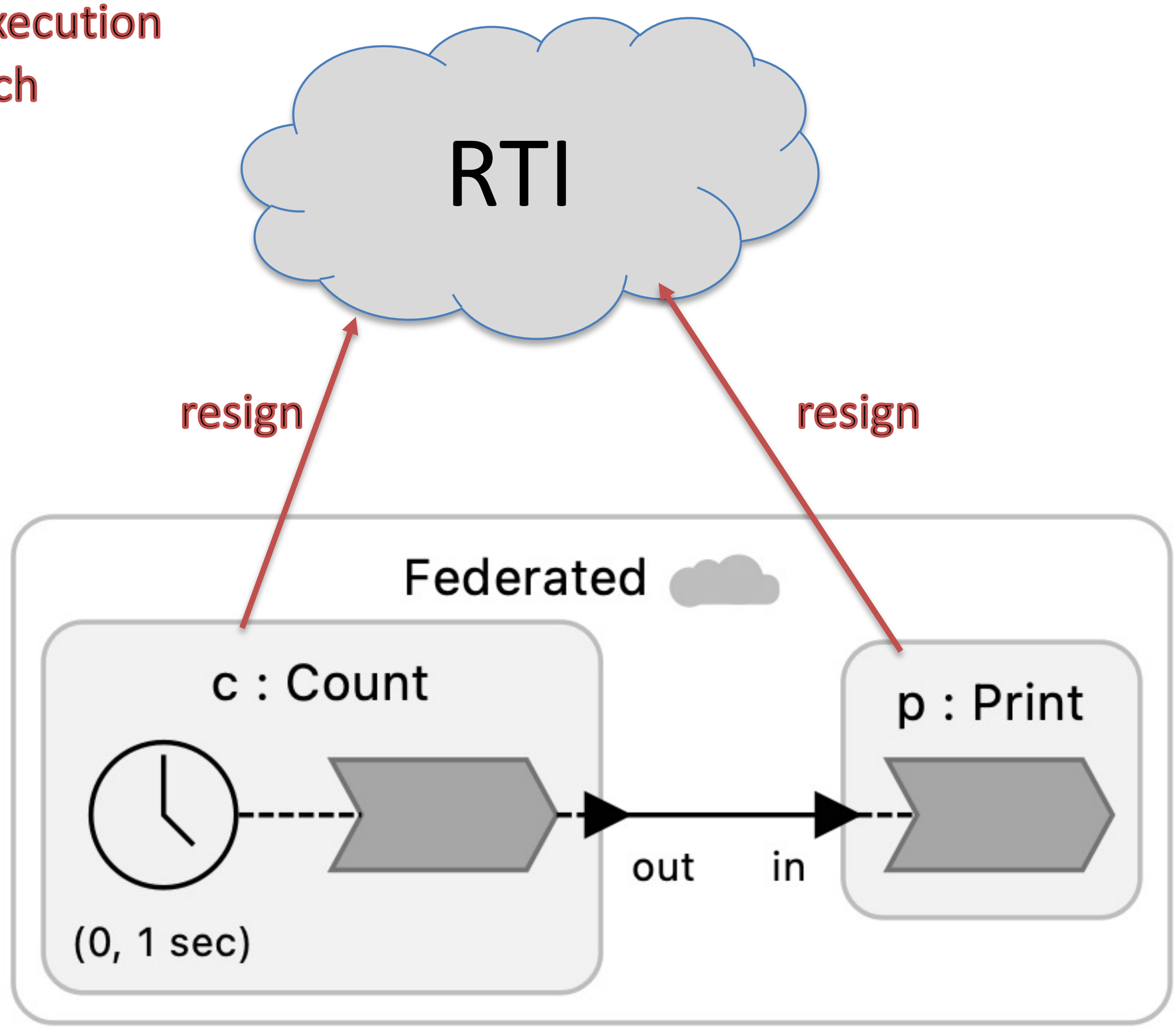


Centralized Coordination: Next Event Request (NET)



Shutdown

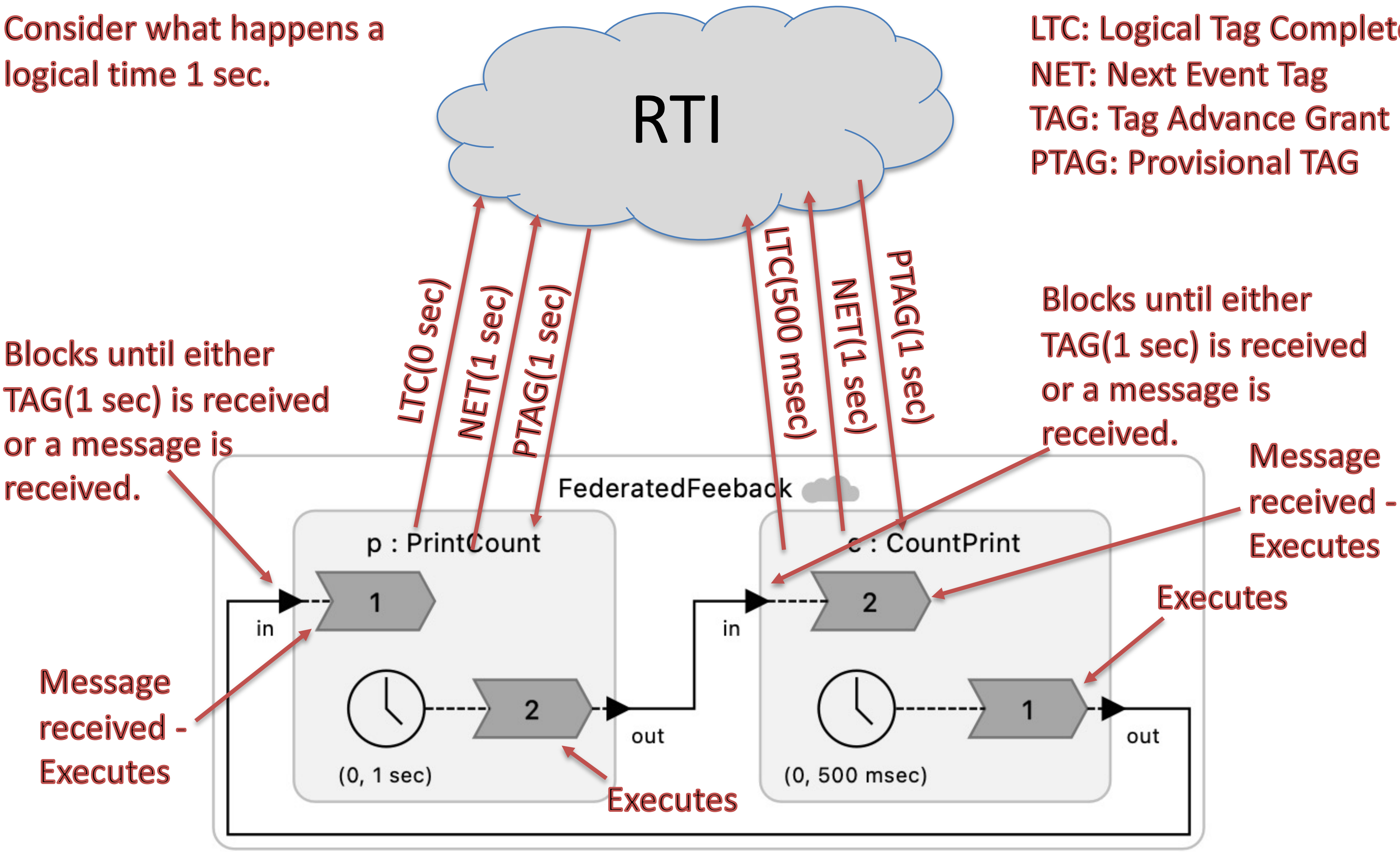
Upon completing execution at timeout time, each federate resigns.



Feedback with Centralized Coordination

Consider what happens a logical time 1 sec.

LTC: Logical Tag Complete
NET: Next Event Tag
TAG: Tag Advance Grant
PTAG: Provisional TAG



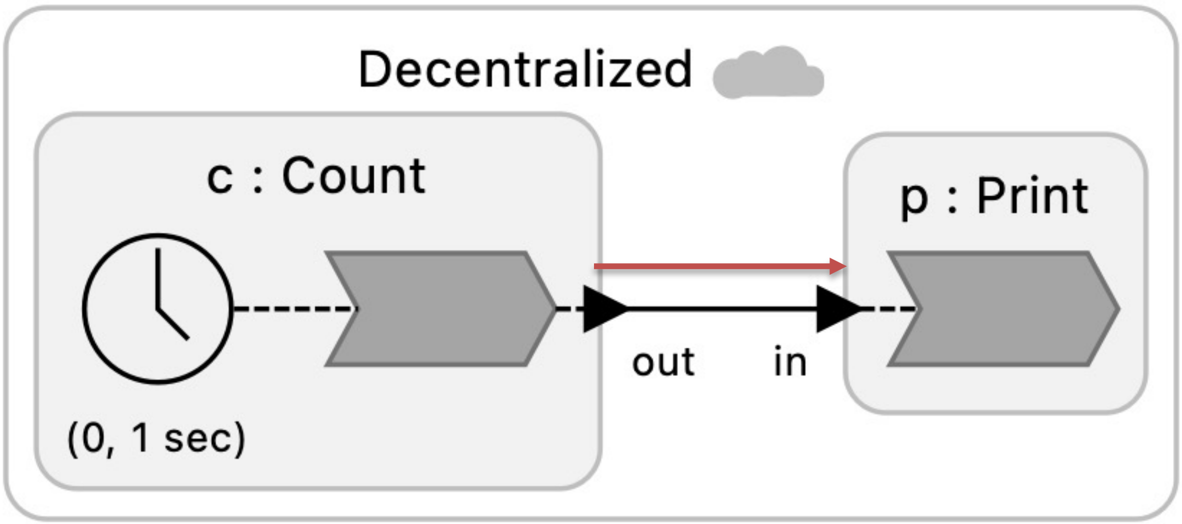
Decentralized Coordination

```
target C {
    coordination: decentralized
};
```

After clock synchronization, establish a direct socket connection and send messages.

RTI not involved after startup.

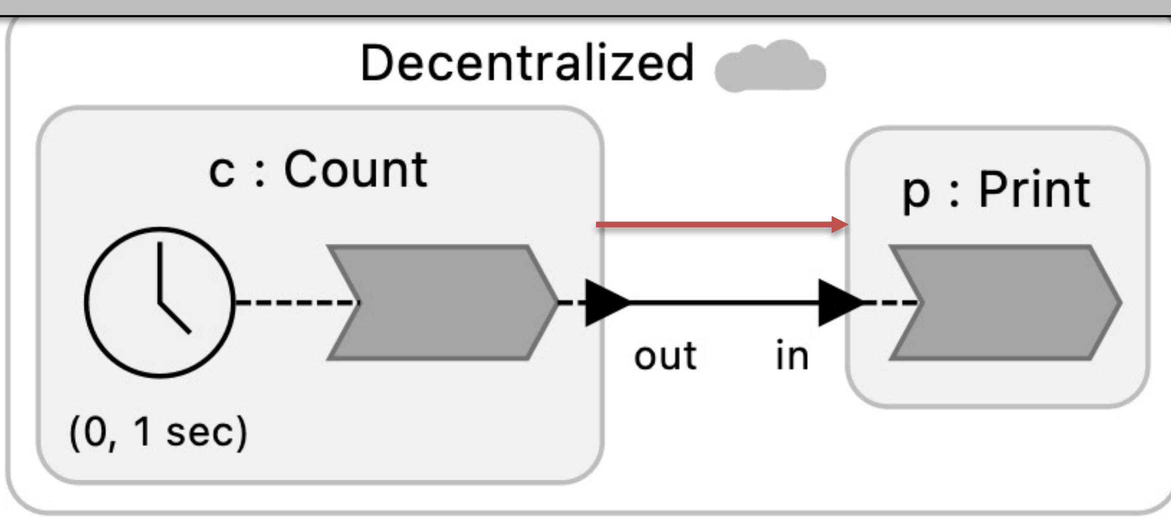
In this example, the default safe-to-process offset of zero mostly works because each federate can safely immediately process an event if it knows about that event.



Decentralized Coordination

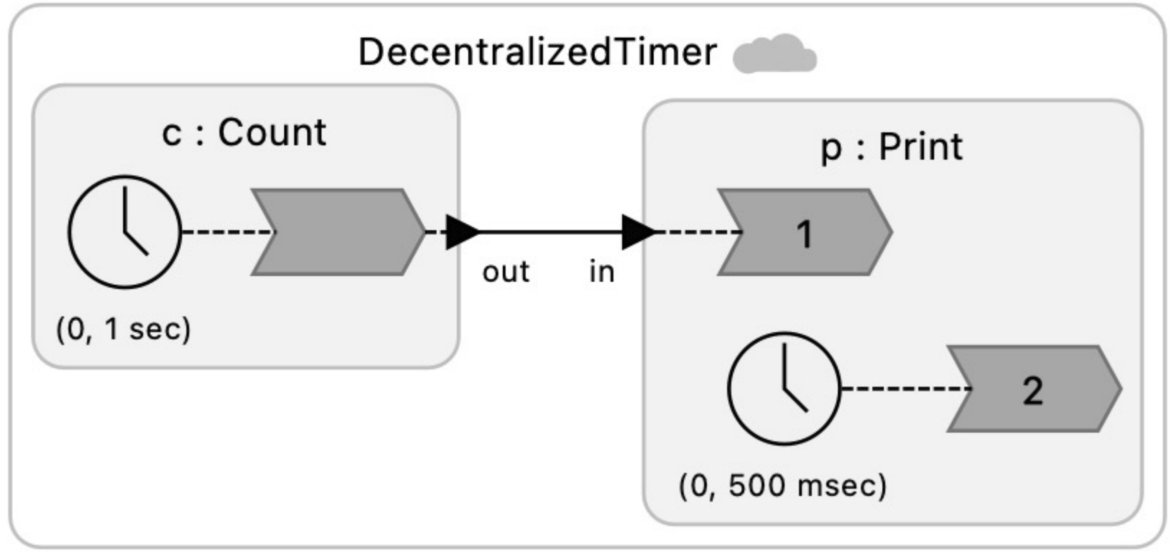
```
EALMAC:c eal$ bin/Decentralized_p
…
Federate 1: ERROR: STP violation occurred in a trigger to reaction 1, and
there is no handler.
**** Invoking reaction at the wrong tag!
Federate 1: Received: 0 at (0, 1)
Federate 1: Received: 1 at (1000000000, 0)
Federate 1: Received: 2 at (2000000000, 0)
Federate 1: Received: 3 at (3000000000, 0)
Federate 1: Received: 4 at (4000000000, 0)
Federate 1: ERROR: Received message too late. Already at stop tag.
Current tag is (5000000000, 0) and intended tag is (5000000000, 0).
Discarding message.
```

What happened?



With Timer

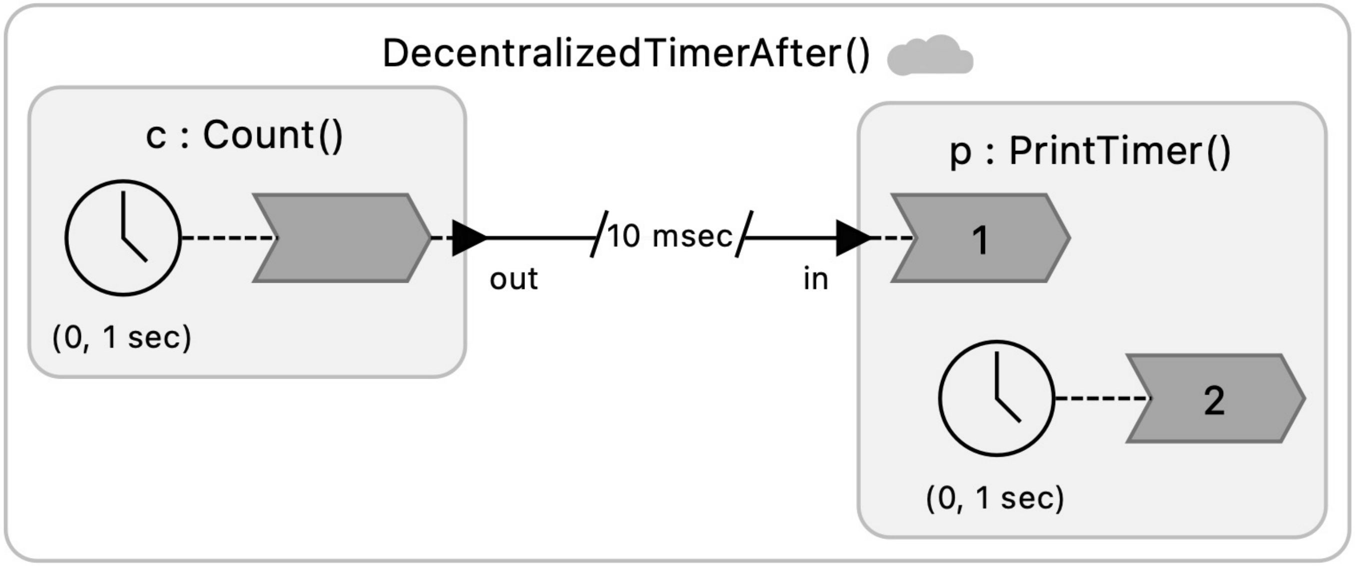
Situation is even worse when destination has timed activity.




```
Federate 1: Starting timestamp is: 1652852093838036000.
Federate 1: Timer ticked at (0, 0).
Federate 1: ERROR: STP violation occurred in a trigger to reaction
1, and there is no handler.
**** Invoking reaction at the wrong tag!
Federate 1: Received: 0 at (0, 1)
Federate 1: Timer ticked at (1000000000, 0).
Federate 1: ERROR: STP violation occurred in a trigger to reaction
1, and there is no handler.
**** Invoking reaction at the wrong tag!
Federate 1: Received: 1 at (1000000000, 1)
Federate 1: Timer ticked at (2000000000, 0).
Federate 1: ERROR: STP violation occurred in a trigger to reaction
1, and there is no handler.
**** Invoking reaction at the wrong tag!
Federate 1: Received: 2 at (2000000000, 1)
Federate 1: Timer ticked at (3000000000, 0).
…
```

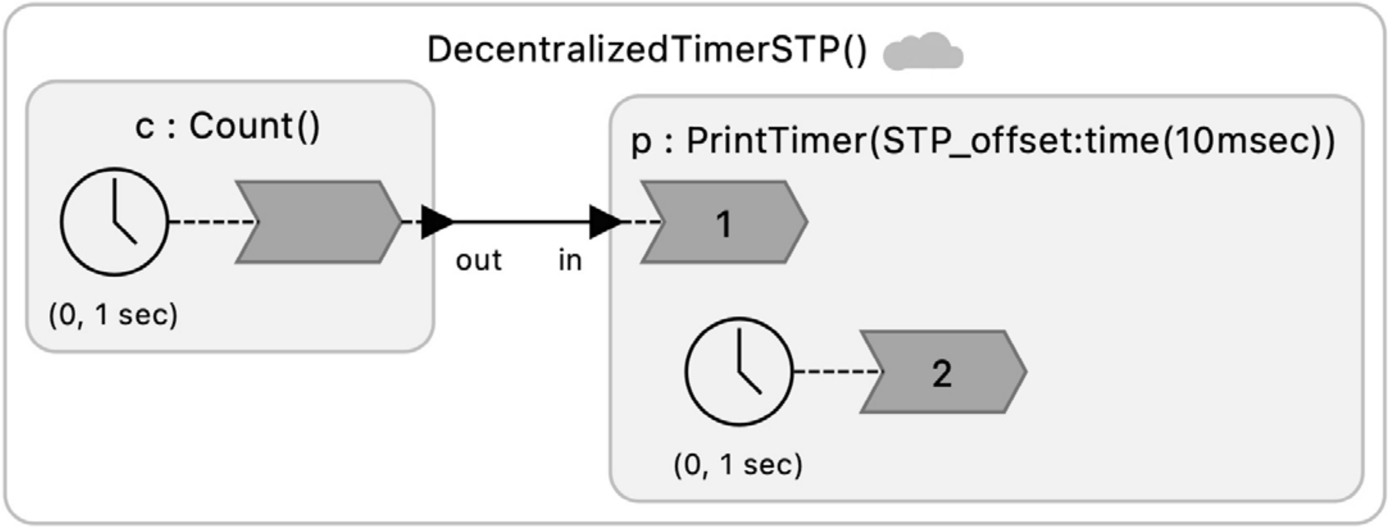
With After

If **after** delay is greater than network delay, then no STP violations occur.




```
Federate 1: Starting timestamp is: 1652852276394596000.
Federate 1: Timer ticked at (0, 0).
Federate 1: Received: 0 at (10000000, 0)
Federate 1: Timer ticked at (1000000000, 0).
Federate 1: Received: 1 at (1010000000, 0)
Federate 1: Timer ticked at (2000000000, 0).
Federate 1: Received: 2 at (2010000000, 0)
Federate 1: Timer ticked at (3000000000, 0).
Federate 1: Received: 3 at (3010000000, 0)
Federate 1: Timer ticked at (4000000000, 0).
Federate 1: Received: 4 at (4010000000, 0)
Federate 1: Timer ticked at (5000000000, 0).
Federate 1 has resigned.
```

With STP Offset

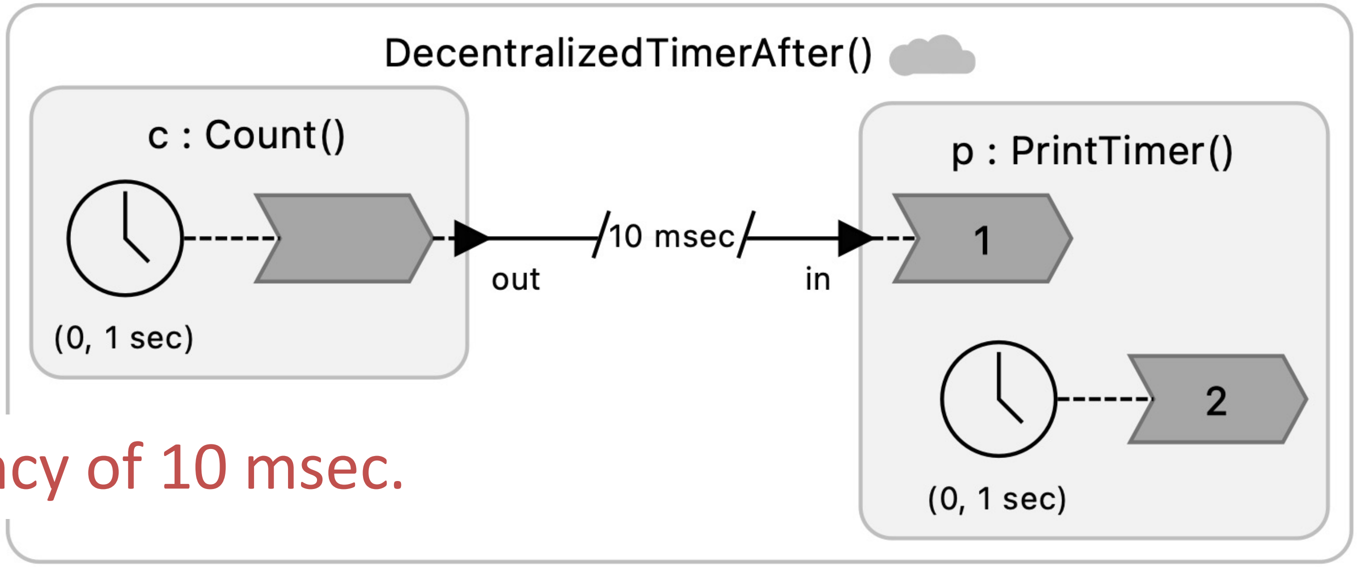


If STP offset is greater than network delay, then no STP violations occur.

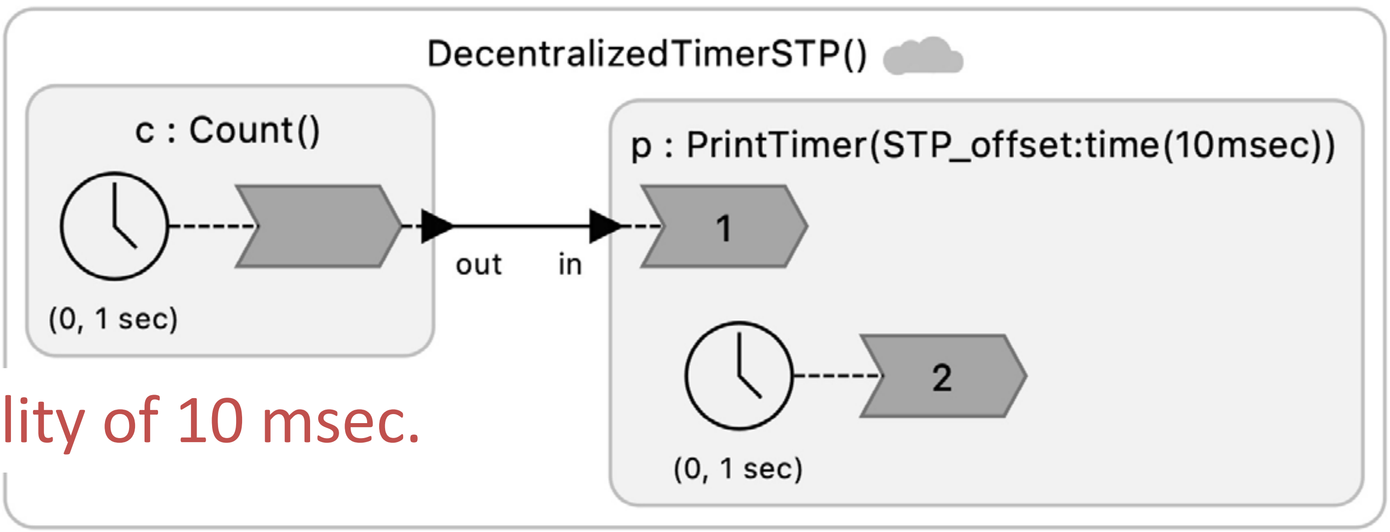
Note: STP_offset here is STA in CAL paper (Safe To Advance).

```
Federate 1: Starting timestamp is: 1652852565042039000.
Federate 1: Received: 0 at (0, 0)
Federate 1: Timer ticked at (0, 0).
Federate 1: Received: 1 at (1000000000, 0)
Federate 1: Timer ticked at (1000000000, 0).
Federate 1: Received: 2 at (2000000000, 0)
Federate 1: Timer ticked at (2000000000, 0).
Federate 1: Received: 3 at (3000000000, 0)
Federate 1: Timer ticked at (3000000000, 0).
Federate 1: Received: 4 at (4000000000, 0)
Federate 1: Timer ticked at (4000000000, 0).
Federate 1: Received: 5 at (5000000000, 0)
Federate 1: Timer ticked at (5000000000, 0).
Federate 1 has resigned.
```

What is the difference?



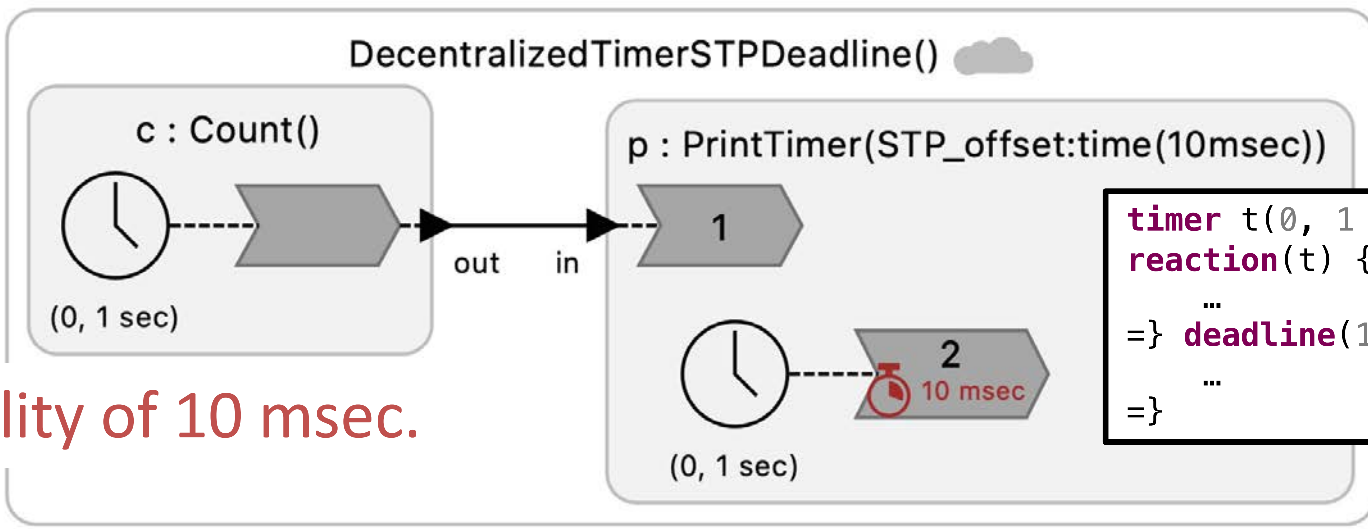
Inconsistency of 10 msec.



Unavailability of 10 msec.

Unavailability

```
Federate 1: Starting timestamp is: 1652853187696038000.
Federate 1: Received: 0 at (0, 0)
Federate 1: **** Deadline violation at (0, 0).
Federate 1: Received: 1 at (1000000000, 0)
Federate 1: **** Deadline violation at (1000000000, 0).
Federate 1: Received: 2 at (2000000000, 0)
Federate 1: **** Deadline violation at (2000000000, 0).
Federate 1: Received: 3 at (3000000000, 0)
…
```




```
timer t(0, 1 sec);
reaction(t) {=
    …
=} deadline(10 msec) {=
    …
=}
```

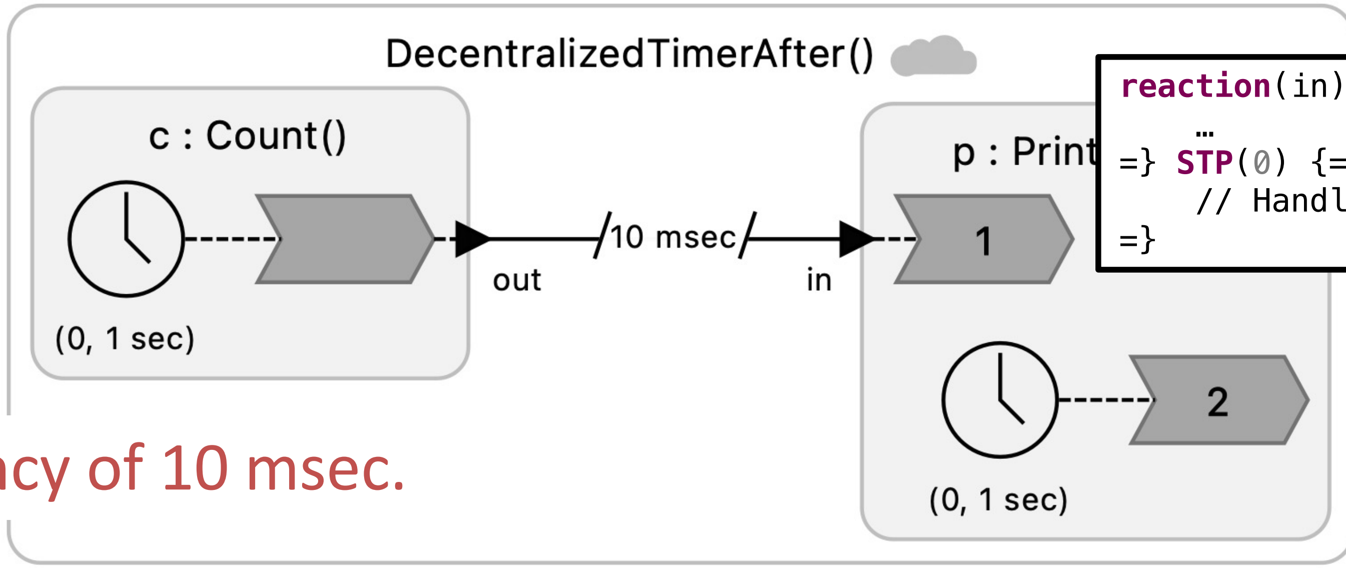
Unavailability of 10 msec.

The CAL Theorem (Preview)

CAL:

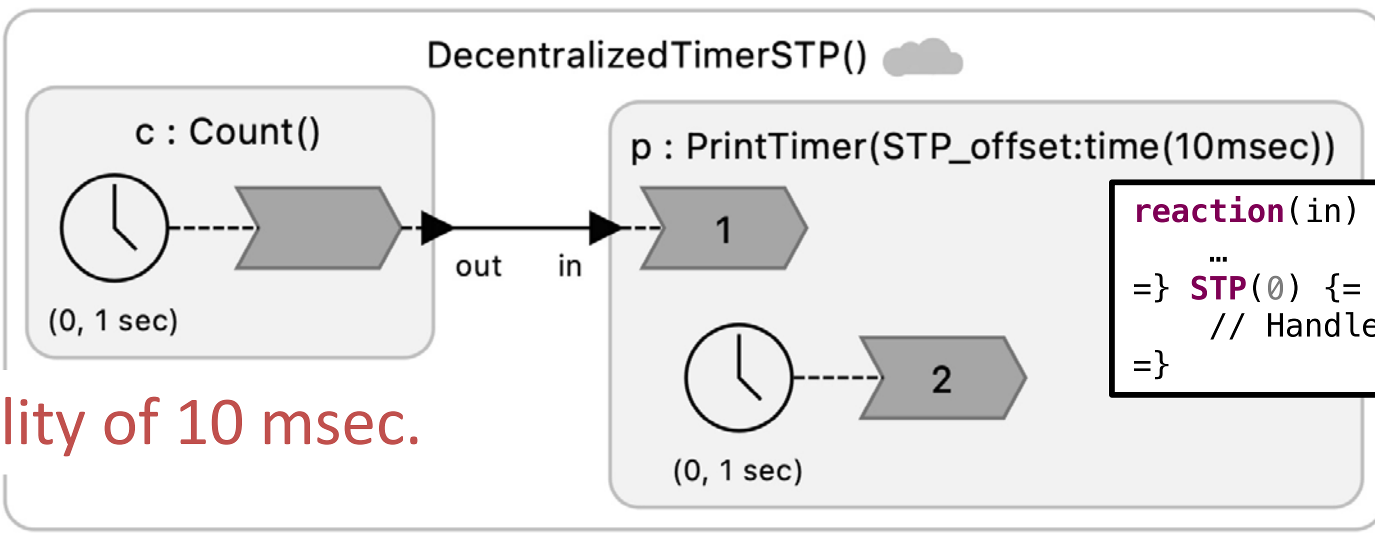
Consistency and/or Availability *must* be sacrificed as network Latency increases in any distributed system.

STP Violation Handlers




```
reaction(in) {=
    …
=} STP(0) {=
    // Handle STP violation
=}
```

Inconsistency of 10 msec.




```
reaction(in) {=
    …
=} STP(0) {=
    // Handle STP violation
=}
```

Unavailability of 10 msec.

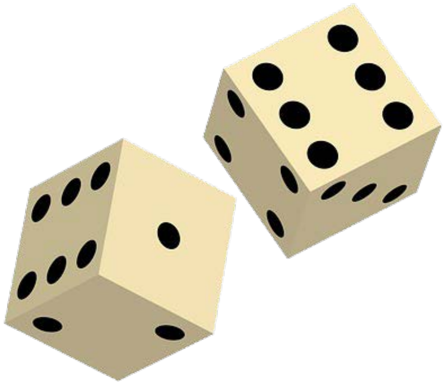
If Assumptions are Met, Determinism!

Assumptions:

- **Deadlines**
  - WCET
  - Schedulability
- **Federated Execution (centralized)**
  - Reliable, in-order network (TCP/IP)
- **Federated Execution (decentralized)**
  - Deadlines
  - Network latency
  - Clock synchronization error

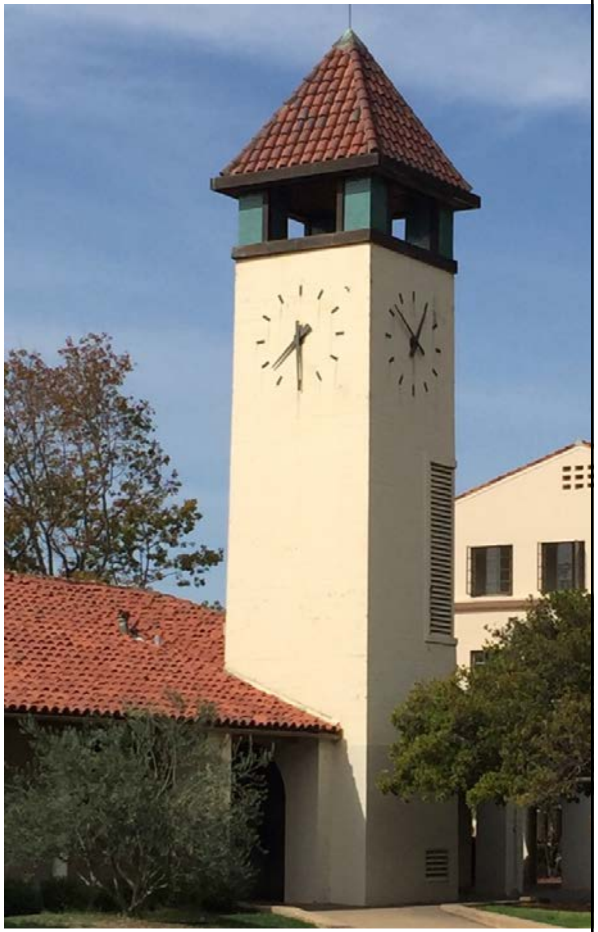
Contrast With…

- Publish and subscribe (e.g. ROS, MQTT)
- Actors (e.g. Akka, Ray)
- Shared memory
- Service-oriented architectures (RPC)



At What Cost Determinism?

- Synchronized clocks
  - These are becoming ubiquitous
- Bounded network latency
  - Violations are *faults*. They are detectable.
- Bounded execution times
  - Only needed in particular places.
  - Solvable with PRET machines (another talk).



Clock Synchronization

- NTP is widely available but not precise enough.
- IEEE 1588 PTP is widely supported in networking hardware but not yet by the OSs.
- Lingua Franca can work without clock synchronization by reassigning timestamps to network messages.
  - In this case, determinism is preserved within each multicore platform, but not across platforms.

Using Synchronized Clocks in Practice

Despite using TCP/IP on Ethernet, this network achieves highly reliable bounded latency.

This Bosch Rexroth printing press is a cyber-physical factory using Ethernet and TCP/IP with **high-precision clock synchronization (IEEE 1588)** on an **isolated LAN**.

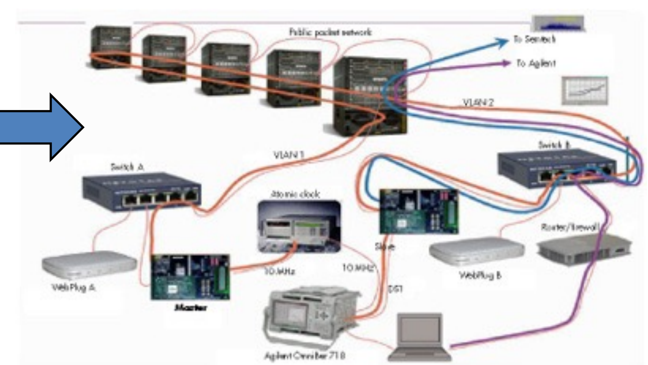
TSN (time-sensitive networks) is starting to become pervasive…

Predicting the Future

Clock synchronization is going to change the world (again)

Gregorian Calendar (BBC history)

Lackawanna Railroad Station, 1907, Hoboken. Photograph by Alicia Dudek



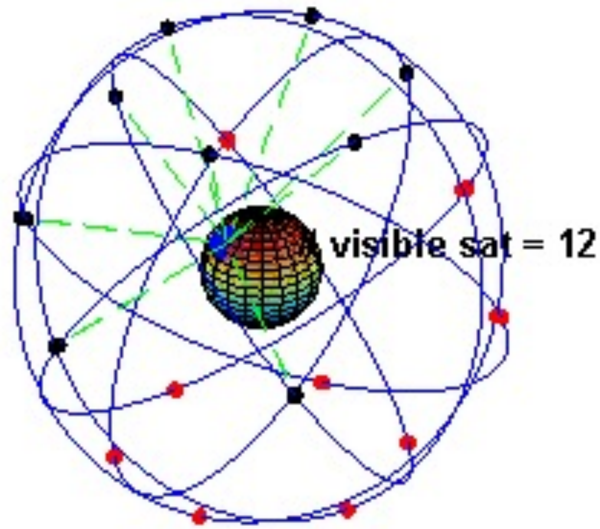
2005: first IEEE 1588 plugfest

1500s
days

1800s
seconds

2000s
nanoseconds

Global Positioning System



Provides ~100ns accuracy to devices with outdoor access.

Precision Time Protocols (PTP) IEEE 1588 on Ethernet

Press Release October 1, 2007

National Semiconductor
The Sight & Sound of Information

NEWS RELEASE

For More Information Contact

Media Contact
Naomi Mitchell
National Semiconductor
(408) 721-2142
naomi.mitchell@nsc.com

Reader Information
Design Support Group
(800) 272-9959
www.national.com

Industry's First Ethernet Transceiver with IEEE 1588 PTP Hardware Support from National Semiconductor Delivers Outstanding Clock Accuracy

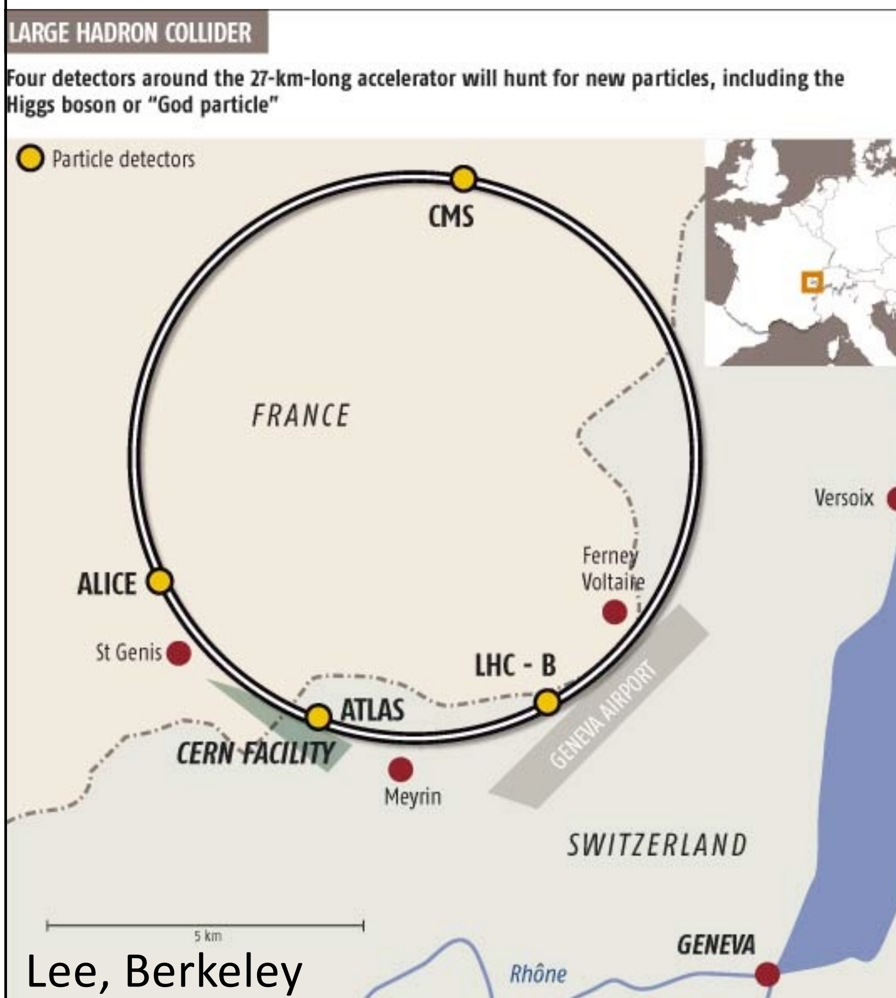
Using DP83640, Designers May Choose Any Microcontroller, FPGA or ASIC to Achieve 8- Nanosecond Precision with Maximum System Flexibility

It is routine for physical network interfaces (PHY) to provide hardware support for PTPs.

With this first generation PHY, clocks on a LAN agree on the current time of day to within ns, far more precise than GPS older techniques like NTP.

An Extreme Example: The Large Hadron Collider

The WhiteRabbit project at CERN is synchronizing the clocks of computers 10 km apart to within 10s of psec using a combination of GPS, IEEE 1588 PTP and synchronous ethernet.



Conclusions

- 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.
- Actors, Pub/Sub, SoA, and shared memory have **none of these properties**.