Global Time

Raj Rajkumar

Lecture #21

** Examples from Prof. Paul Krzyzanowski*

Outline

- Global Time
  - Utility
  - Challenges
  - Terminology
- Event Ordering
  - Logical Clocks
  - Vector Clocks
- Time Synchronization

Utility of Global Time

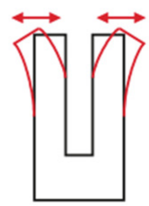
- Event Ordering
  – Global state (fault-tolerance)
  – Database consistency
- Freshness
  – Security
  – Robotic path planning
- Coordinated Actions
  – TDMA networking
- Measurement
  – GPS (signal to distance)

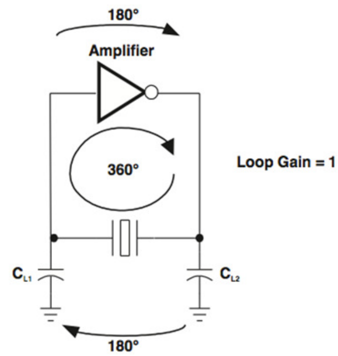
Terms in Synchronized Time

- **Oscillator**
  – Frequency source
- **Clock**
  – Oscillator with counter
- **Skew**
  – Difference between two clocks
- **Timestamp**
  – Measurement of a particular clock value
- **Stability**
  – How well a frequency is maintained
- **Accuracy**
  – Closeness of measurement to (inter)national time
- **Precision**
  – The level of granularity
- **External (Time) Synchronization**
  – Local clocks attempt to synchronize with standard time
- **Internal (Clock) Synchronization**
  – Local clocks attempt to synchronize with each other
  – Imagine only synchronizing frequency / phase

Physical Clock

- A device for time measurement that contains a counter and a physical oscillation mechanism that periodically generates an event to increment a counter
  - Finely-trimmed Quartz Crystal
  - Atomic Clock

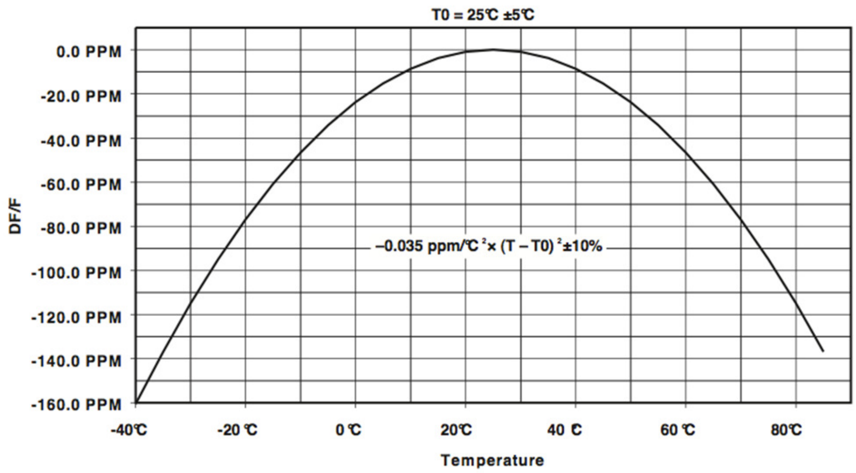




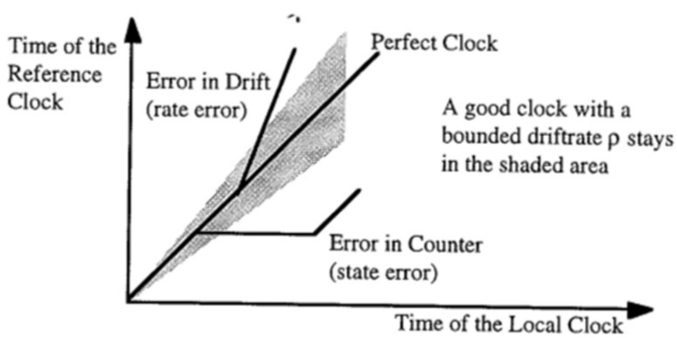
Synchronization Challenges

- Clock Drift
  - Spatially separated devices
- Sampling Jitter
  - Due to measurement error
- Propagation Delay
  - Message time propagation needs to be calculated

Temperature Drift



Drift vs State Error



Sampling Jitter

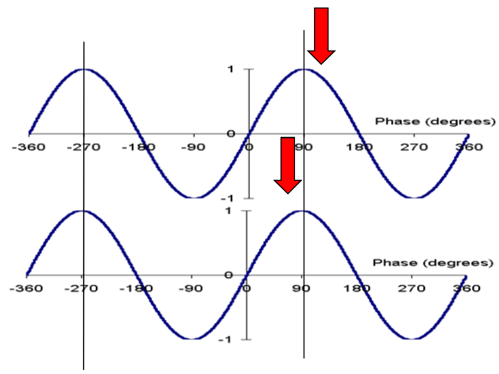
Π Max Precision

ε Message Jitter

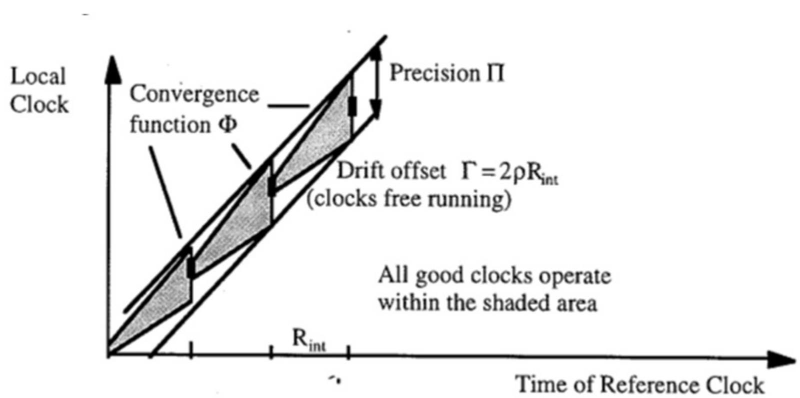
N Number of Devices

$$\Pi = \varepsilon\left(1 - \frac{1}{N}\right)$$

[Lun 84]



State vs Rate Correction



State Correction – Time Jumps

Rate Correction – Adjust the frequency slightly to converge on desired time

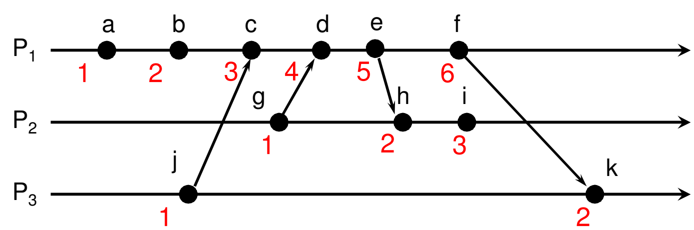
Logical Clocks (Lamport '78)

- **Assign a sequence of numbers to messages**
  - Multiple processes can agree on an ordering of events
  - As compared to the "Time-of-Day" clock
- **Assumptions**
  - No central time source
  - Local free-running clocks
  - Ordering happens only through communications among nodes

"Happened Before" relationship

- ***a* → *b***
  - Event *a* **happened before** event *b*
  - *a* sent a message on one node, *b* received a message on another node
  - *a* occurred on one node, and *b* occurs after *a* on the same node.
- **Transitive Property**
  - If *a* → *b* and *b* → *c*, then *a* → *c*

Event Counting Example



<u>Bad</u> orderings:

e → h

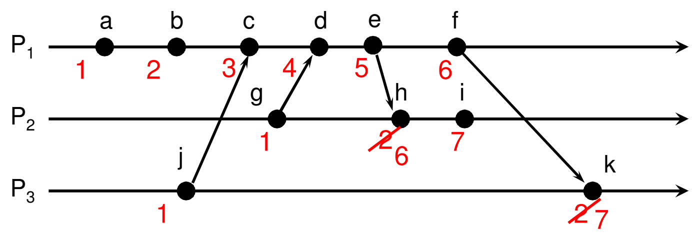
f → k

Logical Clocks

Lamport Timestamp Rules

1. A process increments counter before each event
2. Process messages include its local counter
3. After receiving a message, the receiver sets its counter to **max(received_value + 1, local_counter + 1**)

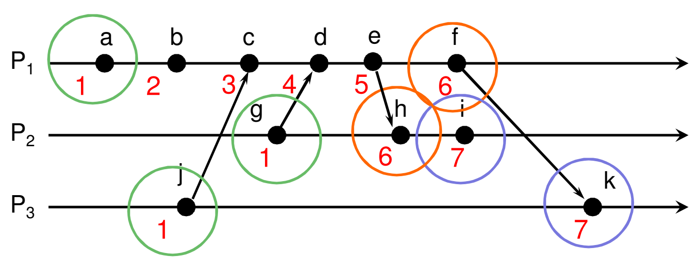
Logical Clocks



Algorithm Provides: "**Partial Ordering**"

- Ordering among related events

Drawback: Identical Timestamps

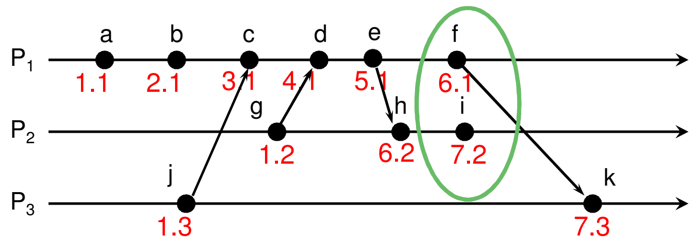


a → *b*, *b* → *c*, …: **local events sequenced**

j → c, *g* → *d*, *e* → *h*, … : Lamport imposes a ***send*** → ***receive*** relationship

Concurrent events (e.g., *a* & *j*) *<u>may</u>* have the same timestamp … or not

Unique (totally ordered) timestamps



Still no way to determine which events are *causally* related.

Problem: Detecting causal relations

If Lamport(*e*) < Lamport(*e*')

– Cannot conclude that *e* caused *e*'

That is, looking at Lamport timestamps,

– Cannot conclude which events are causally related

Solution: use a **vector clock**

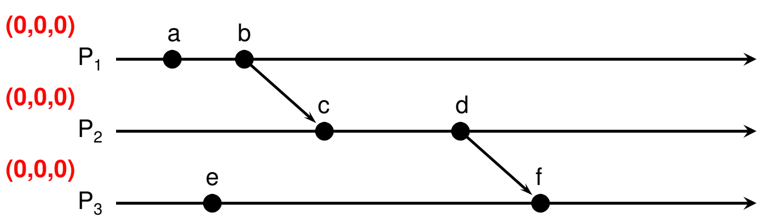
Vector Clocks

Rules:

1. Vector initialized to 0 at each process
   $V_i[j] = 0$ for $i, j = 1, \ldots, N$
2. Process increments its element of the vector in local vector before time-stamping event:
   $V_i[i] = V_i[i] + 1$
3. Message is sent from process P_i with V_i attached to it
4. When P_j receives message, it compares vectors element by element and sets its local vector to the higher of the two values
   $V_j[i] = \max(V_i[i], V_j[i])$ for $i=1, \ldots, N$

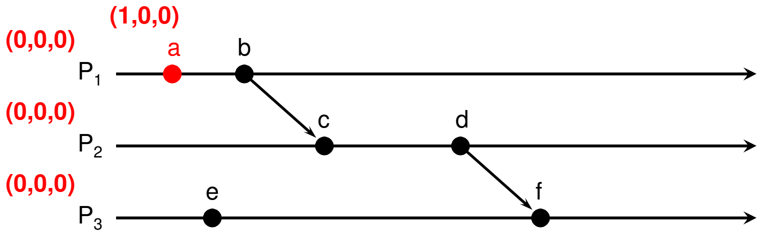
Carnegie Mellon 18-648: Embedded Real-Time Systems Electrical & Computer ENGINEERING

Vector timestamps



Carnegie Mellon 18-648: Embedded Real-Time Systems Electrical & Computer ENGINEERING

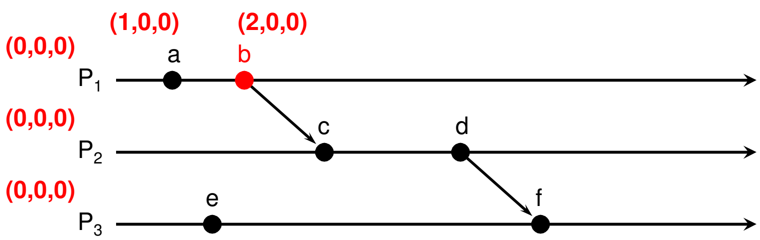
Vector timestamps



Event	timestamp
a	(1,0,0)

Carnegie Mellon 18-648: Embedded Real-Time Systems Electrical & Computer ENGINEERING

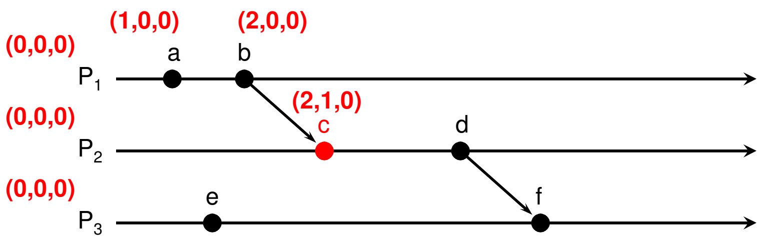
Vector timestamps



Event	timestamp
a	(1,0,0)
b	(2,0,0)

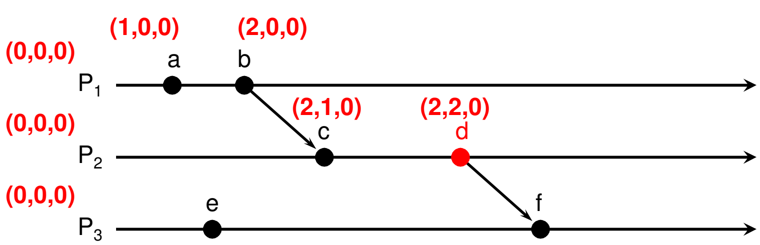
Carnegie Mellon 18-648: Embedded Real-Time Systems Electrical & Computer ENGINEERING

Vector timestamps



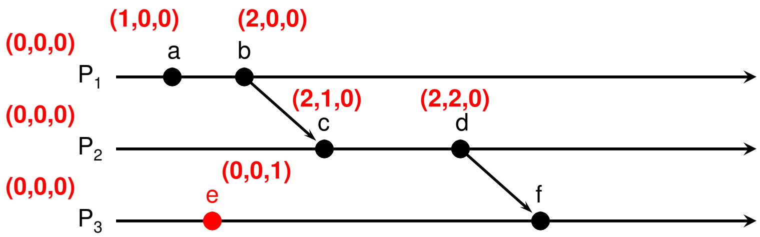
Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)

Vector timestamps



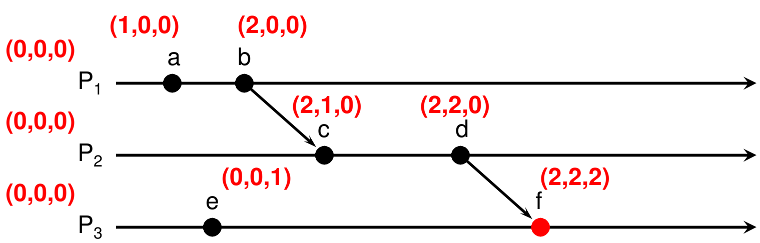
Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)

Vector timestamps



Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)
e	(0,0,1)

Vector timestamps



Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)
e	(0,0,1)
f	(2,2,2)

Comparing vector timestamps

Define

$V = V'$ iff $V[i] = V'[i]$ for $i = 1 \ldots N$

$V \leq V'$ iff $V[i] \leq V'[i]$ for $i = 1 \ldots N$

$V < V'$ iff $V[i] < V'[i]$ for $i = 1 \ldots N$

$V \geq V'$ iff $V[i] \geq V'[i]$ for $i = 1 \ldots N$

$V > V'$ iff $V[i] > V'[i]$ for $i = 1 \ldots N$

Vector comparison

For any two events e and e',

if $e \rightarrow e'$ then $V(e) < V(e')$ (just like Lamport's clock)

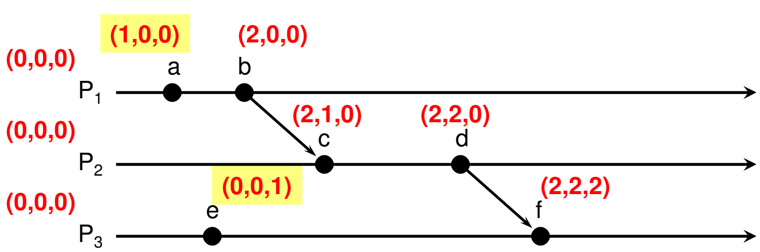
if $\mathbf{V(e) < V(e')}$ **then** $\mathbf{e \rightarrow e'}$

Two events are said to be **concurrent** if neither

$V(e) \leq V(e')$ **nor**

$V(e') \leq V(e)$

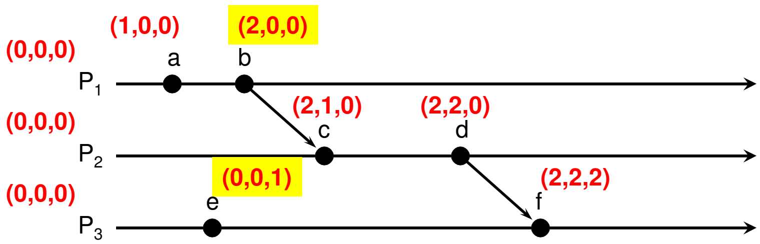
Vector timestamps



Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)
e	(0,0,1)
f	(2,2,2)

concurrent events

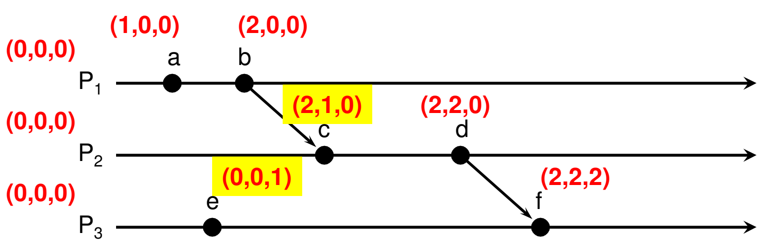
Vector timestamps



Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)
e	(0,0,1)
f	(2,2,2)

concurrent events

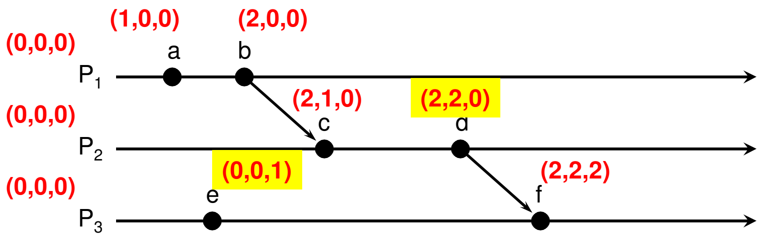
Vector timestamps



Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)
e	(0,0,1)
f	(2,2,2)

concurrent events

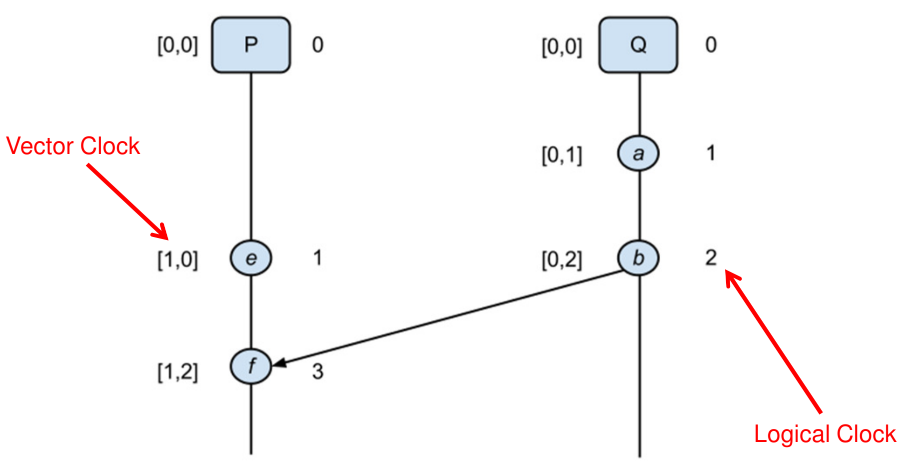
Vector timestamps



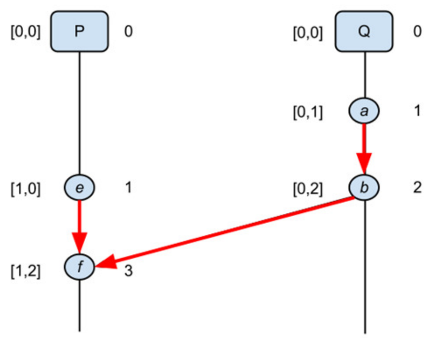
Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)
e	(0,0,1)
f	(2,2,2)

concurrent events

Logical *vs* Vector (1 of 3)



Logical vs Vector (2 of 3)



Lamport Clock
- *a before b*
- *a before f*
- *b before f*
- *e before f*

Vector Clock
- *a* before *b* AND *b* not before *a*
- *b* before *f* AND *f* not before *b*
- ***e* not before *b***
  AND *b* not before *e*

Red line indicates *"happened before"* relationship

Logical vs Vector (3 of 3)

Suppose that

C(*X*) is the Lamport timestamp of *X*

VC(*Y*) is the Vector timestamp of *Y*

- **Lamport timestamps tell us:**
  - *X* not before *Y*, if C(*X*) ≥ C(*Y*)
- **Vector clocks can tell us that:**
  - *X* not before *Y* if VC(*X*) > VC(*Y*)
  - *X* before *Y* if VC(*X*) < VC(*Y*)
  - Neither *X* before *Y*, nor *Y* before *X* otherwise (concurrent)

Time Synchronization

- **Time synchronization is critical in *many* system layers**
  - Beam-forming, localization, distributed DSP
  - Data aggregation & caching
  - TDMA guard bands
  - "Traditional" uses (debugging, user interaction…)

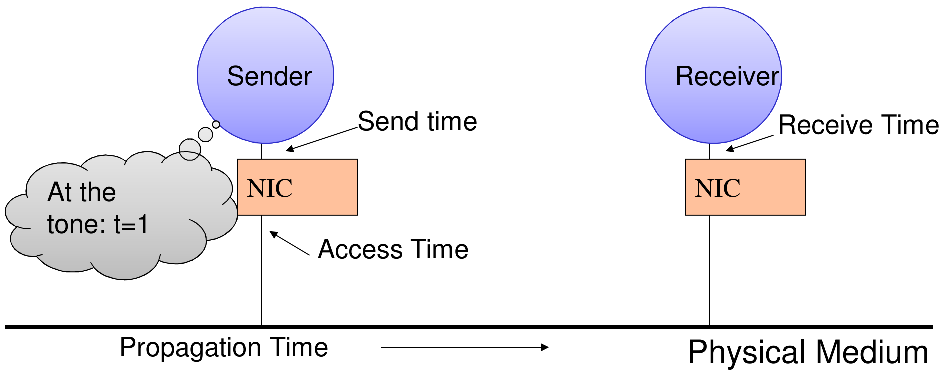
- **But time-synchronization needs are non-uniform**
  - Maximum Error
  - Lifetime
  - Scope & Availability
  - Efficiency (use of power and time)
  - Cost and form factor

Time Synchronization Approaches

- **Message Passing**
  - Exchanging time stamps between devices
- **Global Broadcasts**
  - Transmit a signal over a large area
- **Common Observations**
  - Receive existing signal
  - Quasar Pulses, Quantum-Entangled Particles

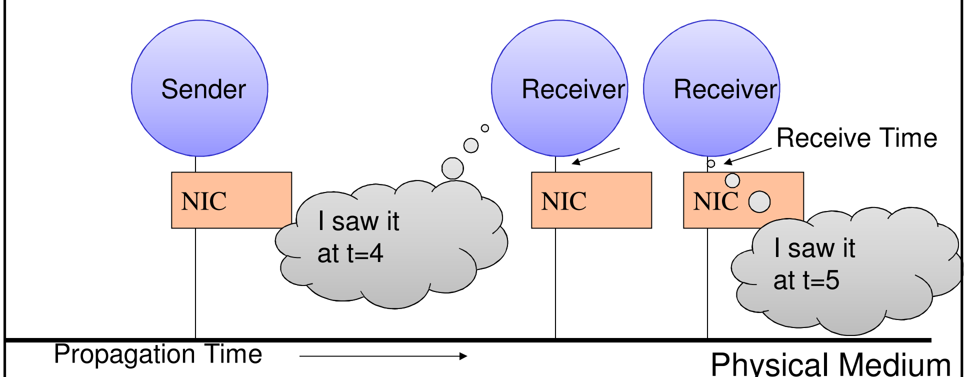
Traditional sync

Problem: Many sources of unknown, non-deterministic latency between timestamp and its reception

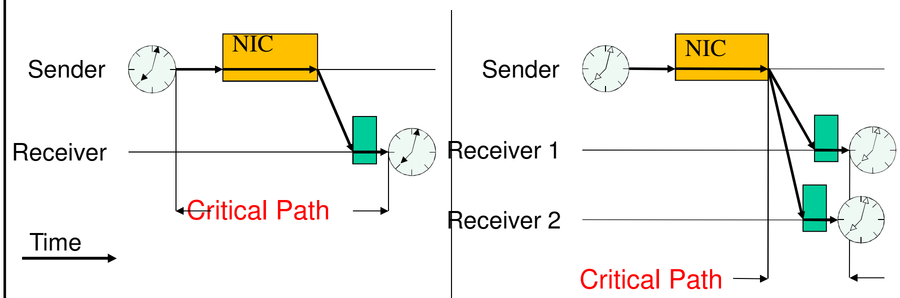


Reference Broadcasts

Synchronize receivers *with one another*, NOT sender with receiver



RBS reduces error by removing much of it from the critical path



<u>Traditional critical path</u>: From the time the sender reads its clock, to when the receiver reads its clock

<u>RBS</u>: Only sensitive to the differences in receive time and propagation delay

Summary

- **Causality**
  - If *a*→*b*, then event *a* can affect event *b*
- **Concurrency**
  - If neither *a* → *b* nor *b* → *a*, then one event cannot affect the other
- **Partial Ordering**
  - Causal events are sequenced
- **Total Ordering**
  - All events are sequenced