# CS 3551 DISTRIBUTED COMPUTING

# UNIT II

# LOGICAL TIME AND GLOBAL STATE

Logical Time: Physical Clock Synchronization: NTP – A Framework for a System of Logical Clocks – Scalar Time – Vector Time; Message Ordering and Group Communication: Message Ordering Paradigms – Asynchronous Execution with Synchronous Communication – Synchronous Program Order on Asynchronous System – Group Communication – Causal Order – Total Order; Global State and Snapshot Recording Algorithms: Introduction – System Model and Definitions – Snapshot Algorithms for FIFO Channels

## Logical Time

### Definition

A system of logical clocks consists of a time domain T and a logical clock C. Elements of T form a partially ordered set over a relation <. This relation is usually called the happened before or causal precedence. Intuitively, this relation is analogous to the earlier than relation provided by the physical time. The logical clock C is a function that maps an event e in a distributed system to an element in the time domain T, denoted as C(*e*) and called the timestamp of *e*, and is defined as follows:

$$C : H \mapsto T,$$

such that the following property is satisfied: for two events $e_i$ and $e_j$ ,
$e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$. This monotonicity property is called the clock consistency condition.

When T and C satisfy the following condition, for two events

$$e_i \text{ and } e_j,\ e_i \rightarrow e_j \Leftrightarrow C(e_i) < C(e_j),$$

the system of clocks is said to be strongly consistent.

### Implementing logical clocks

Implementation of logical clocks requires addressing two issues: data structures local to every process to represent logical time and a protocol (set of rules) to update the data structures to ensure the consistency condition.

Each process $p_i$ maintains data structures that allow it the following two capabilities:

1. A *local logical clock*, denoted by $lc_i$, that helps process $p_i$ measure its own progress.
2. A *logical global clock*, denoted by $gc_i$, that is a representation of process $p_i$'s local view of the logical global time. It allows this process to assign consistent timestamps to its local events. Typically, $lc_i$ is a part of $gc_i$.

The protocol ensures that a process's logical clock, and thus its view of the global time, is managed consistently. The protocol consists of the following two rules:

1. R1 This rule governs how the local logical clock is updated by a process when it executes an event (send, receive, or internal).
2. R2 This rule governs how a process updates its global logical clock to update its view of the global time and global progress. It dictates what information about the logical time is piggybacked in a message and how this information is used by the receiving process to update its view of the global time.

**Scalar time**

**Definition:**

The scalar time representation was proposed by Lamport in 1978 as an attempt to totally order events in a distributed system. Time domain in this representation is the set of non-negative integers. The logical local clock of a process pi and its local view of the global time are squashed into one integer variable $C_i$.

Rules **R1** and **R2** to update the clocks are as follows:

1. **R1** Before executing an event (send, receive, or internal), process $p_i$ executes the following:

$$C_i := C_i + d \qquad (d > 0)$$

In general, every time **R1** is executed, *d* can have a different value, and this value may be application-dependent. However, typically *d* is kept at 1 because this is able to identify the time of each event uniquely at a process, while keeping the rate of increase of *d* to its lowest level.

2. **R2** Each message piggybacks the clock value of its sender at sending time. When a process $p_i$ receives a message with timestamp $C_{msg}$, it executes the following actions:

   1. $C_i := max(C_i, C_{msg},);$
   2. execute **R1**;
   3. deliver the message.

Figure shows the evolution of scalar time with d=1.

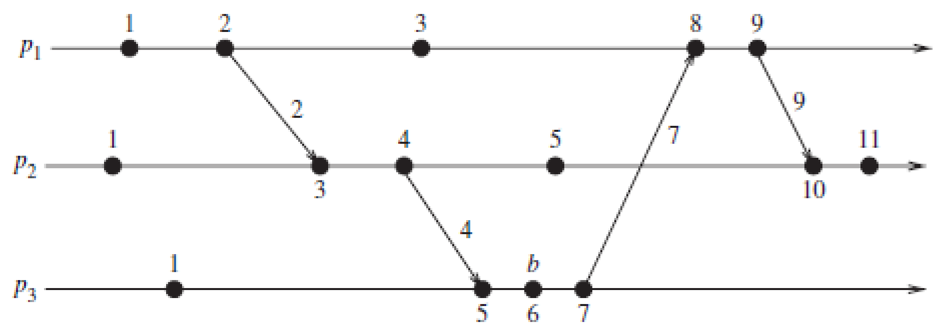


Figure: Evolution of scalar time

**Basic properties**

**Consistency property**

Clearly, scalar clocks satisfy the monotonicity and hence the consistency property:

for two events $e_i$ and $e_j$, $e_i \rightarrow e_j \Longrightarrow C(e_i) < C(e_j)$.

**Total Ordering**

Scalar clocks can be used to totally order events in a distributed system. The main problem in totally ordering events is that two or more events at different processes may have an identical timestamp.

**Event counting**

If the increment value d is always 1, the scalar time has the following interesting property: if event e has a timestamp h, then h−1 represents the minimum logical duration, counted in units of events, required before producing the event e; we call it the height of the event e.

**No strong consistency**

The system of scalar clocks is not strongly consistent; that is, for two events $e_i$ and $e_j$,

$C(e_i) < C(e_j) \not\Rightarrow e_i \rightarrow e_j$