Event Ordering
The two critical differences between centralized and distributed systems are:

- **absence of shared memory**
- **absence of a global clock**

We will study:

- **how programming mechanisms change as a result of these differences**
- **algorithms that operate in the absence of a global clock**
- **algorithms that create a sense of a shared, global time**
- **algorithms that capture a consistent state of a system in the absence of shared memory**
Event Ordering

How can the events on P be related to the events on Q?

Which events of P “happened before” which events of Q?

Partial answer: events on P and Q are strictly ordered. So:

\[ P_1 \rightarrow P_2 \rightarrow P_3 \]

and

\[ Q_1 \rightarrow Q_2 \rightarrow Q_3 \]
Event Ordering

Realization: the only events on P that can causally affect events on Q are those that involve communication between P and Q.

If $P_1$ is a send event and $Q_2$ is the corresponding receive event then it must be the case that:

$$P_1 \rightarrow Q_2$$
**Event Ordering**

"Happened Before" relation:
If $E_i$ and $E_j$ are two events of the same process, then
$$E_i \rightarrow E_j \text{ if } i < j.$$  
If $E_i$ and $E_j$ are two events of different processes, then
$$E_i \rightarrow E_j$$
if $E_i$ is a message send event and $E_j$ is the corresponding message receive event.

The relation is transitive.
Lamport's Algorithm

Lamport's algorithm is based on two implementation rules that define how each process's local clock is incremented.

Notation:

- the processes are named $P_i$,
- each process has a local clock, $C_i$
- the clock time for an event $a$ on process $P_i$ is denoted by $C_i(a)$.

Rule 1:

If $a$ and $b$ are two successive events in $P_i$ and $a \rightarrow b$ then $C_i(b) = C_i(a) + d$ where $d > 0$.

Rule 2:

If $a$ is a message send event on $P_i$ and $b$ is the message receive event on $P_j$ then:

- the message is assigned the timestamp $t_m = C_i(a)$
- $C_j(b) = \max(C_j, t_m + d)$
Example of Lamport’s Algorithm

P1

1 2 3 4 5 6 7 8 9 10

P2

1 2 3 4 5 6 7 8 9 10

P3

1 2 3 4 5 6 7 8 9 10
Limitation of Lamport's Algorithm

In Lamport's algorithm two events that are causally related will be related through their clock times. That is:

If \( a \rightarrow\rightarrow b \) then \( C(a) < C(b) \)

However, the clock times alone do not reveal which events are causally related. That is, if \( C(a) < C(b) \) then it is not known if \( a \rightarrow\rightarrow b \) or not. All that is known is:

if \( C(a) < C(b) \) then \( b \rightarrow\rightarrow a \)

It would be useful to have a stronger property - one that guarantees that

\( a \rightarrow\rightarrow b \) iff \( C(a) < C(b) \)

This property is guaranteed by Vector Clocks.
Vector Clock Rules

Each process $P_i$ is equipped with a clock $C_i$ which is an integer vector of length $n$.

$C_i(a)$ is referred to as the timestamp event $a$ at $P_i$.

$C_i[i]$, the $i$th entry of $C_i$ corresponds to $P_i$’s on logical time.

$C_i[j], j \neq i$ is $P_i$’s best guess of the logical time at $P_j$

Implementation rules for vector clocks:

[IR1] Clock $C_i$ is incremented between any two successive events in process $P_i$

$$C_i[i] := C_i[i] + d \quad (d > 0)$$

[IR2] If event $a$ is the sending of the message $m$ by process $P_j$, then message $m$ is assigned a vector timestamp $t_m = C_i(a)$; on receiving the same message $m$ by process $P_j$, $C_j$ is updated as follows:

$$\forall k, C_j[k] := \max(C_j[k], t_m[k])$$
Event Ordering

Vector Clocks

\[ \begin{align*}
P_1 &: (1,0,0) \\
P_2 &: (0,1,0) \\
P_3 &: (0,0,1)
\end{align*} \]
Causal Ordering of Messages

Send($M_1$)

P1

Send($M_2$)

P2

P3

Space

Time

Virginia Tech
Birman-Schiper-Stephenson Protocol

1. Before broadcasting a message \( m \), a process \( P_i \) increments the vector time \( VT_{P_i}[i] \) and timestamps \( m \). Note that \((VT_{P_i}[i] - 1)\) indicates how many messages from \( P_i \) precede \( m \).

2. A process \( P_j \neq P_i \), upon receiving message \( m \) timestamped \( VT_m \) from \( P_i \), delays its delivery until both the following conditions are satisfied.
   a. \( VT_{P_j}[i] = VT_m[i] - 1 \)
   b. \( VT_{P_j}[k] \geq VT_m[k] \quad \forall k \in \{1,2,\ldots,n\} - \{i\} \)

   where \( n \) is the total number of processes.

   Delayed messages are queued at each process in a queue that is sorted by vector time of the messages. Concurrent messages are ordered by the time of their receipt.

3. When a message is delivered at a process \( P_j \), \( VT_{P_j} \) is updated according to the vector clocks rule [IR2]